Escuela Técnica Superior de Ingeniería del Diseño

Universitat Politècnica de València

Trabajo Fin de Grado

Ingeniería en Electrónica Industrial y Automática

DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR PROGRAMABLE DE PATRONES DE MOVIMIENTO PARA MOTORES DE CORRIENTE CONTINUA CON APLICACIÓN EN ROBOTS Y MÁQUINAS CNC

Documentos:

1. Memoria
2. Planos
3. Pliego de condiciones
4. Presupuesto
5. Manual de usuario
6. Anexos:
    - 6-1: Documentación código
    - 6-2: Listados código
    - 6-3: Hojas de datos

Autor:
D. José Félix González Rojo

Tutor:
D. Ángel Perles Ivars

Cotutor:
D. Miguel Sánchez López

Valencia, julio de 2016

Gracias a Angel Perles Ivars y Miguel Sánchez López

por indicarme el camino y ayudarme a recorrerlo.

A Antonio J. Sánchez Salmerón por su ayuda.

A mi família y mi pareja por apoyarme.

Resumen

En los últimos años, ha habido un gran aumento de los productos de consumo basado en procesadores ARM. Este incremento ha sido motivado por la eficiencia y el bajo costo de estos procesadores, que los hacen adecuados para dispositivos de baja potencia y aplicaciones que no hacen un uso intensivo de la CPU. Por otro lado, la robótica y la industria actual se basa en el uso del motor de corriente continua con escobillas como accionamiento y actuador por excelencia para cualquier proceso de producción. El presente trabajo busca examinar la viabilidad de implementar un controlador discreto programable en un ARM para controlar la posición, velocidad y par de un motor de corriente continua industrial que pueda tener aplicación en robots o cualquier máquina CNC.

Este trabajo también tiene la intención de servir como una referencia general para la implementación de controladores discretos abarcando desde la obtención del modelo del proceso (teórica y experimentalmente) hasta la programación y configuración del microcontrolador, pasando por el diseño de distintos tipos de controles y la generación de trayectorias.

En concreto, el trabajo realizado implementa un control de posición y velocidad en una STM32F429-DISCOVERY, para un motor Pittman DC040B-2 con encoder incremental y se efectua el seguimiento de trayectorias generadas en un solo eje mediante una plataforma de ensayo y pruebas diseñada para tal efecto.

Palabras clave: Sistema embebido, ARM Cortex-M, control discreto, generación y seguimiento de trayectorias, motor CC, adquisición de datos.

Summary

During the last years the amount of ARM processor based products in use has increased. This raise has been motivated by the high eficiency and the low cost of its processors, which make them appropiate for low power devices and aplications that do not need an intense CPU use. In addition, present robotic and current industry is based on brushed DC motor as main drive for every production process. The aim of this project is to test the viability of a discrete controller implementation on an ARM Device to control position, speed and torque of an industrial DC motor that could be used on any CNC Machine or robot.

This document has also the intention to serve as a general reference for discrete controller implementation, from process model obtaining (theorically and experimentally) to setup and programing of the microcontroller, through different controllers design and trayectory generation.

Concretly, the project implements a position and speed control on a STM32F429-DISCOVERY, for a Pittman DC040B-2 motor with an incremental encoder and it makes generated trayectories tracking in just one axis using a test platform designed and developped for this purpose.

Key words: embebed system, ARM Cortex-M, discrete control, trayectory generation and tracking, DC motor, data adquisition.

Resum

En els últims anys, hi ha hagut un gran augment dels productes de consum basat en processador ARM. Este increment ha sigut motivat per l'eficiència i el baix cost d'estos processadors, que els fan adequats per a dispositius de baixa potència i aplicacions que no fan un ús intensiu de la CPU. D'altra banda, la robòtica i la indústria actual es basa en l'ús del motor de corrent continu amb graneretes com a accionament i actuador per excel·lència per a qualsevol procés de producció. El present treball busca examinar la viabilitat d'implementar un controlador discret programable en un ARM per a controlar la posició, velocitat i parell d'un motor de corrent continu industrial que puga tindre aplicació en robots o qualsevol màquina CNC.

Este treball també té la intenció de servir com una referència general per a la implementació de controladors discrets comprenent des de l'obtenció del model del procés (teòrica i experimentalment) fins a la programació i configuració del microcontrolador, passant pel disseny de distints tipus de controls i la generació de trajectòries.

En concret, el treball realitzat implementa un control de posició i velocitat en una STM32F429-DISCOVERY, per a un motor Pittman DC040B-2 amb *encoder* incremental i s'efectua el seguiment de trajectòries generades en un sol eix per mitjà d'una plataforma d'assaig i proves dissenyada per a tal efecte.

Paraules clau: Sistema embegut, ARM Cortex-M, control discret, generació i seguiment de trajectòries, motor CC, adquisició de dades.

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR PROGRAMABLE DE PATRONES DE MOVIMIENTO PARA MOTORES DE CORRIENTE CONTINUA CON APLICACIÓN EN ROBOTS Y MÁQUINAS CNC

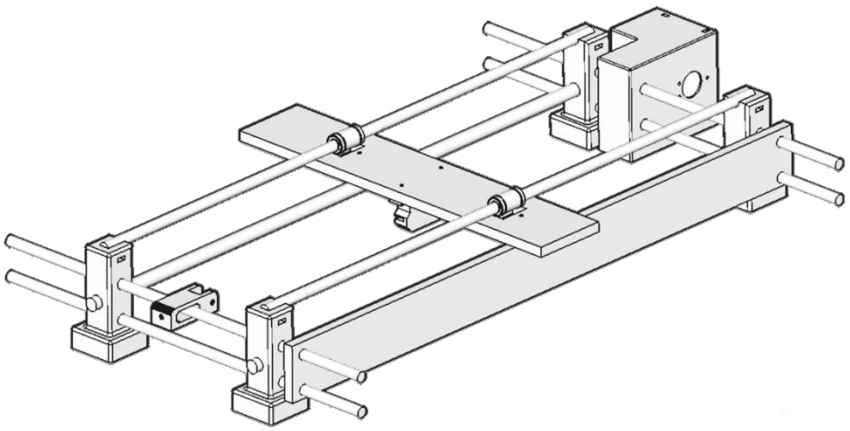
1.MEMORIA

Autor:
D. José Félix González Rojo

Tutor:
D. Ángel Perles Ivars

Cotutor:
D. Miguel Sánchez López

Valencia, julio de 2016



Contenido:

1. Objeto 3
2. Antecedentes 3
3. Estudio de necesidades, factores a considerar: limitaciones y condicionantes 4
3.1. Control analógico y control discreto 4
3.2. Muestreo 5
3.3. Aspectos prácticos de la implementación 7
3.4. Modelado del sistema 9
3.5. Sensorización 11
3.6. Sistema de adquisición de datos 11
3.7. Acción de control 12
3.8. Generación de referencias 13
4. Planteamiento de soluciones alternativas y justificación de la solución adoptada 14
4.1 Tipos de muestreo 14
4.1.1 Muestreo por polling 14
4.1.2 Muestreo por retraso software 15
4.1.3 Muestreo por interrupción 15
4.1.4 Muestreo por Acceso Directo a Memoria (DMA) 15
4.1.5 Solución adoptada 16
4.2 Realizaciones del controlador 16
4.2.1 Forma directa 16
4.2.2 Forma estándar 17
4.2.3 Forma serie 18
4.2.4 Forma paralela 18
4.2.5 Forma de escalera 19
4.3 Modelos del proceso 20
4.3.1 Modelado matemático 20
4.3.2 Modelado no paramétrico. Respuesta ante escalón 28
4.3.3 Modelado paramétrico. Estimación mediante aproximación por mínimos cuadrados 47
4.4 Elección del sensor 51
4.4.1 Medida de la posición 51
4.4.2 Medida de la velocidad 53
4.4.3 Medida de la intensidad 54
4.5 Elección del sistema de adquisición de datos 54
4.5.1 Tarjetas de adquisición de datos y PC 54
4.5.2 Lego Mindstorms NXT 55

4.5.2 Microchip PIC 56
4.5.3 Arduino 56
4.5.4 ST STM32F4 56
4.6 Elección de la acción de control 57
4.6.1 Conversión D/A. Acción de control directa bipolar 57
4.6.2 Modulación de ancho de pulso (PWM) 58
4.7 Control del proceso 60
4.7.1 Controladores proporcionales, derivativos e integrales (PID) 60
4.7.2 Espacio de estados. Control por realimentación del estado observado 63
4.7.3 Control por asignación de polos 66
4.7.4 Control por cancelación. Control en tiempo finito y tiempo mínimo 71
4.7.5 Efecto de la perturbación 77
4.8 Generación de trayectorias y patrones de movimiento 78
4.8.1 Perfil de movimiento trapezoidal 80
4.8.2 Perfil de movimiento de curva en S. Aproximación de tercer orden 81
4.8.3. Perfil de movimiento de curva en S. Aproximación de cuarto orden y orden n 82
4.8.4. Perfil de movimiento senoidal 83
5. Descripción detallada de la solución adoptada 84
5.1 Sistema de trabajo 84
5.2 Índices de rendimiento 86
5.3 Implementación 88
5.3.1 Disposición y conexiones del hardware 88
5.3.1 Configuración de la tarjeta 89
5.3.2 Implementación del patrón de movimiento 97
5.3.3. Diseño de los controladores a implementar. PID 99
5.3.3. Resultados. Implementación del control y realización de movimientos punto a punto . 117
6. Conclusiones 123
6.1. Sobre el trabajo realizado 123
6.2. Futuras mejoras 124
7. Bibliografía 125

1. Objeto

El objeto del presente proyecto es el desarrollo e implementación de un control en bucle cerrado de un motor de corriente continua para el seguimiento de patrones de movimiento programables, para ello:

- Se evaluará el comportamiento de distintos controles, con diferentes configuraciones de referencia.

- Se utilizarán distintas configuraciones de muestreo para facilitar el trabajo del microprocesador a la hora de tomar y procesar los datos.

- Se utilizará un microcontrolador de bajo coste, con alta disponibilidad y por tratarse de un controlador compatible con la mayoría de entornos y de filosofía de código abierto; que además es muy utilizado como solución de multitud de productos comerciales.

La aplicación del proyecto se enfocará a sistemas robotizados y máquinas de control numérico por computador (CNC), para la planificación de patrones de movimiento y trayectorias con un *feedback* de datos.

2. Antecedentes

En la actualidad los actuadores de los sistemas robotizados y máquinas de control numérico por computador (CNC) están formados, en su inmensa mayoría, por motores paso a paso (PAP) y servomotores con control embebido. Esto es debido a la necesidad de precisión en los movimientos para el posicionamiento de los sistemas y a la importancia de la concordancia de la actuación de los distintos elementos para generar las trayectorias deseadas.

En igualdad de condiciones respecto a estos elementos mencionados, es decir, a la misma tensión de alimentación, con las mismas dimensiones, etc. los motores de corriente continua ofrecen características similares o incluso superiores a un precio más reducido. El inconveniente para su uso es el control complejo y poco preciso que se implementa.

Existen, no obstante, a nivel industrial, controladores muy precisos para motores de corriente continua desarrollados, producidos y comercializados por diferentes empresas tales como Performance Motion Devices Corporation (PMD corp) o AMS. Sin embargo el precio de estos productos es elevado y su contenido cerrado a terceros y consumidores, lo que impide el desarrollo por la comunidad de este tipo de configuraciones que podrían resultar, de otra forma, sencillas y económicas.

Por otro lado, la implementación de los actuadores antes nombrados se lleva a cabo en lazo abierto a partir de código G de instrucciones. Esta implementación es susceptible de errores externos al control y la programación, tanto mecánicos (atascos, obstáculos, roturas,...) como eléctricos (desconexiones, sobrecargas,…) sin exceptuar accidentes. El desarrollo e implementación de un control en bucle cerrado que transformara estos motores actuadores en motores inteligentes, capaces de conocer las condiciones en las que actúan y con un sistema embebido que pueda tomar decisiones en función de la variación de las mismas conllevaría una mejora no sólo en la precisión, calidad y fiabilidad de las máquinas, sino en la seguridad.

De la unión de estas dos ideas surge la motivación del presente proyecto.

3. Estudio de necesidades, factores a considerar: limitaciones y condicionantes

En el entorno industrial actual cualquier empresa que se dedique a producción o montaje depende del trabajo de varias máquinas de control numérico o distintas configuraciones de robots, que realizan labores de precisión y que requieren de un esfuerzo superior al que una persona podría realizar. El empleo de estos sistemas supone un aumento de la productividad (si bien supone una importante inversión económica) y, lo que es más importante, la seguridad.

Las principales necesidades para estos sistemas son, como ya se ha indicado previamente, la precisión en los movimientos, la fiabilidad y calidad de la implementación, y la seguridad. Una de las posibles soluciones al sistema supone el desarrollo de un controlador programable completo para el actuador, lo que podríamos llamar en nuestro caso, un motor inteligente. Este sería capaz de realizar el seguimiento de trayectorias mediante patrones de movimiento preprogramados con un control en bucle cerrado de posición, velocidad y par, pudiendo realizar el movimiento y detectar cambios en el funcionamiento que permitan actuar con seguridad ante situaciones no esperadas.

La consecución de este controlador conlleva el análisis de una serie de condicionantes que estudiaremos a continuación.

3.1. Control analógico y control discreto

Cuando se pretende realizar el control de un proceso físico como supone una máquina eléctrica se dispone de varias posibilidades. Una de las más utilizadas es la realimentación de la salida en bucle cerrado ya que proporciona mayor rapidez en el control, permite eliminar errores de modelado, eliminar el efecto de perturbaciones, etc.

Clásicamente se ha venido utilizando un control analógico, por lo que el regulador se diseñaba e implementaba en el plano continuo, mediante amplificadores operacionales, resistencias, condensadores, etc.

La tendencia actual a la hora de implementar un control para un actuador como un motor es el uso de la tecnología digital. Esto es debido, entre otras razones, a:

- La disminución del precio de los equipos informáticos.
- La independencia del controlador respecto al hardware, pudiendo utilizarse un mismo microcontrolador para distintos actuadores.
- La posibilidad de implementar algoritmos complejos.
- La posibilidad de comunicación con otros dispositivos y almacenamiento de datos, siendo más sencilla la implementación de sistemas de prevención de fallos y permitiendo la automatización de múltiples procesos.

Además, el mismo computador que realiza el control sirve de monitor y/o supervisor del mismo, pudiendo implementarse interfaces más amigables con el usuario y pudiendo modificar los programas de manera sencilla. Es por estos motivos por lo que nos decidiremos por una implementación digital en el presente trabajo.

Sin embargo, este nuevo abanico de posibilidades presenta también algunos aspectos a tener en cuenta: conceptos como el muestreo, la conversión, los retenedores,…

Así, si distinguimos entre el diagrama básico de un control analógico y uno digital en bucle cerrado, básicamente, las acciones que realiza el computador son adquisición o muestreo, cálculo de la acción de control, mantenimiento o espera, y salida; como podemos ver en las figuras 1 y 2.

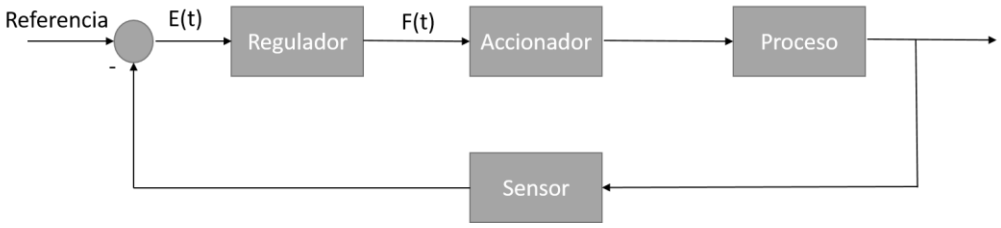


Figura 1. Diagrama de control analógico en bucle cerrado.

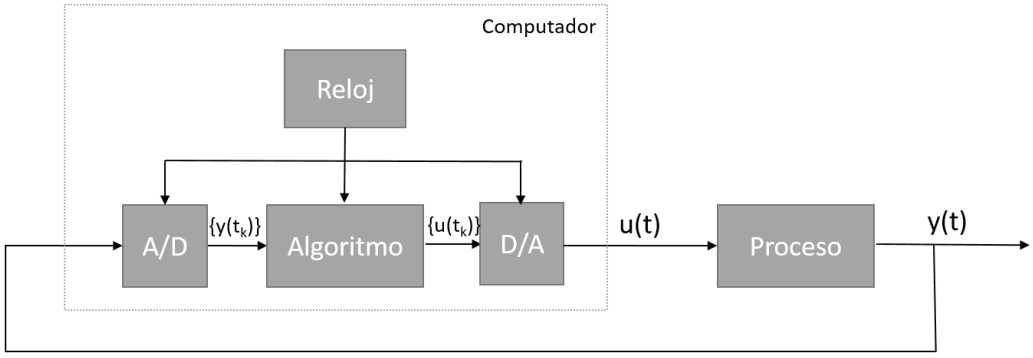


Figura 2. Diagrama de control digital en bucle cerrado.

Podemos decir que la estructura general del algoritmo de cualquier controlador en un dispositivo basado en procesador es:

Inicialización de variables
Bucle temporizado según periodo de muestreo T
    Medir salida
    Calcular acción de control
    Aplicar acción de control
    Actualizar variables
Fin del bucle

Así pues, vamos a exponer algunos conceptos claves relativos a estas acciones.

3.2. Muestreo

Entendemos por muestreo la medición de señales que se convierten en valores digitales que puede entender el procesador.

El tiempo entre dos instantes de muestreo es lo que denominamos periodo de muestreo. Normalmente optaremos por muestreos periódicos para un tiempo que decidiremos suficiente, sin embargo, existen otras opciones como muestreos multifrecuencia (variando el periodo de muestreo en distintas iteraciones del bucle o en distintos bucles); o muestreos basados en eventos o interrupciones (que detectan variaciones en un umbral de la salida para medir la señal, utilizados en

conversores A/D o en encoders).

Si tenemos un periodo de muestreo suficientemente pequeño, virtualmente el control discreto tendrá el mismo comportamiento que un control continuo.

Uno de los problemas asociados a este concepto es el aliasing, que es un error relativo al muestreo y su frecuencia que resulta en la obtención de unas señales con frecuencias medidas distintas a las reales.

Partiremos del cumplimiento del teorema de Shannon-Nyquist para evitar este fenómeno. El teorema dice así:

"*Dada la frecuencia de corte wc de una señal x(t) y dada la frecuencia del muestreo ws con que queremos muestrear dicha señal, si se cumple que:*

$$wc < \frac{ws}{2}$$

Ecuación 1. Formulación del Teorema de Shannon-Nyquist.

Se verifica que la señal muestreada x(t) caracteriza por completo a la señal continua x(t) y es posible reconstruir esta última con exactitud a partir de la muestreada, mediante un filtro pasa-banda ideal .*"

Pese a todo esto existe una problemática que nos abordará durante todo el trabajo y es la diferencia entre el modelo, calculado o medido, y el proceso real [MAR 15]. Además, nuestro sistema de control ideal mide, calcula y aplica la acción de control en el mismo instante, mientras que el procesador real requiere más tiempo.

Para realizar nuestra solución, en primer lugar procederemos a desarrollar la implementación del periodo de muestreo en el punto 4.1. Pero, anticipándonos a la obtención del modelo, podemos definir el periodo de muestreo que utilizaremos en la implementación del proyecto.

Empezaremos estableciendo los límites del periodo de muestreo a implementar. El límite superior, el máximo periodo de muestreo que podemos utilizar, es el que indica el teorema de Shannon-Nyquist.

El menor periodo de muestreo lo establece el hardware, tanto el proceso como el sistema de adquisición y control.

Para un control en bucle cerrado algunos criterios para establecer un periodo de muestreo son:

- El tiempo de respuesta $T < \frac{t_e}{10}$ Siendo t_e el tiempo de establecimiento.

- La pulsación natural: $\omega_s > 10\ \omega_n$, normalmente $10\ \omega_n\ \leq\ \omega_s \leq 30\ \omega_n$.

- El ancho de banda: $\omega_s > 10\ \omega_b$, habitualmente $10\ \omega_b\ \leq\ \omega_s \leq 20\ \omega_b$

El criterio que seguiremos será:

$$T < \frac{2\pi}{15\sqrt{\sigma^2 + \omega_p^2}}$$

Ecuación 2. Criterio para la elección del periodo de muestreo.

Pudiéndose determinar a partir de los polos deseados en bucle cerrado, si los conociéramos, o a partir de una respuesta experimental conociendo:

Tiempo de establecimiento: $t_e = 4\tau = \frac{4}{\sigma} \rightarrow \sigma$

Tiempo de pico: $t_p = \frac{\pi}{\omega_p} \rightarrow \omega_p$

Sobreoscilación: $\delta = e^{-\sigma \cdot t_p} \cdot 100 \rightarrow \sigma$

El periodo de muestreo que utilizaremos para nuestro hardware (con un tiempo de establecimiento de 0.1s y una sobreoscilación de un 5%) será:

$$T < \frac{2\pi}{15\sqrt{\sigma^2 + \omega_p^2}}$$

$$T = \frac{2 * \pi}{15\sqrt{40^2 + 0.075^2}} = 0.0157$$

Ecuación 3. Elección del periodo de muestreo.

Por lo que es suficiente cualquier valor menor a 0,0157. Nosotros tomaremos, por ser un tiempo ya comprobado en el estudio de sistemas similares y por comodidad a la hora de la implementación, 0.01, es decir, 10 ms.

Una vez tratada la problemática fundamental del muestreo y decidido el periodo que se utilizará para desarrollo del control, vamos a comentar aspectos prácticos a tener en cuenta a la hora de la implementación de un controlador programable.

3.3. Aspectos prácticos de la implementación

Un controlador programable implica necesariamente una realización discreta, lo que conlleva una serie de diferencias respecto a la realización analógica convencional ya que, pese a trabajar con una capacidad de cálculo elevada en un entorno con elementos cuantificados, el proceso a controlar sigue siendo analógico.

Un aspecto de implementación a tener en cuenta es la diferencia entre las señales analógicas que entran y salen del proceso, y las señales discretas que entran y salen del controlador. Dejaremos la adquisición y aplicación de estas señales para un poco más adelante, pero debemos tener en cuenta que la conversión D/A y A/D realizada en el procesador para el muestreo de la salida y la aplicación de la acción de control conlleva errores debidos a la cuantificación de los sistemas digitales, pero esta cuantificación depende de la realización del controlador discreto que implementemos. Por ejemplo, para la expresión general del controlador discreto en representación

externa:

$$G_R(z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_m z^{-m}}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_n z^{-n}}$$

Ecuación 4. Controlador discreto en representación externa.

Podemos elegir varias implementaciones, como la forma directa (ecuación en diferencias), la forma estándar (que reduce el número de variables a implementar), la forma serie (o de Jordan), la forma paralela y la forma en escalera. Estas tres últimas implementaciones descomponen la función de transferencia en funciones de primer y segundo orden, lo que disminuye los errores de truncado. Todas ellas se exponen más adelante en el apartado 4.2. De esta forma podremos expresar el regulador discreto en formas que el microcontrolador pueda manejar y entender.

Así pues, necesitamos un regulador que realizar con una de estas implementaciones. En general, a la hora de calcular el regulador tenemos dos formas de proceder:

- Calcular primero el regulador a partir del modelo $G_P(s)$ en el plano continuo $G_R(s)$ (lugar de las raíces en el plano S, teoremas del módulo y argumento, etc.) y obtener el equivalente discreto $G_R(z)$.

- Obtener primero el modelo discreto del proceso $G_P(z)$ y diseñar un regulador discreto para este modelo.

Como se puede apreciar, en ambos casos, partimos de un modelo matemático que tiene las mismas características que el proceso a controlar.

Otro aspecto práctico a considerar es la saturación de la acción de control y el efecto *windup* que llevan asociados los controladores que implementen un integrador debido a la no linealidad de los actuadores. Utilizaremos una condición por software para evitarlo.

Los actuadores, por lo general, tienen un comportamiento más o menos lineal entre un valor máximo y mínimo. En muchas ocasiones, por el diseño de los controladores, se llevan las acciones de control fuera de este rango lineal. Esto produce la saturación del controlador y del actuador. Cuando esto ocurre la acción de control calculada no es la acción aplicada pues el actuador no es capaz de realizarla.

Debemos, por tanto, revisar nuestro planteamiento y entender la ley de control como un sistema dinámico que tendrá una realización distinta en función de la elección de las variables de estado. Tendremos que asumir que nuestro muestreo y acción de control nunca serán reales pues requeriría de una precisión del microcontrolador infinita.

Para el caso de un controlador con acción integral la saturación produce *windup* en el integrador, que sigue aumentando el valor de la acción integral aun cuando la saturación se produce y la acción de control no se alcanza por el actuador. Esto significa que, si en algún momento el error fuera pequeño, la acción de control todavía sería demasiado grande.

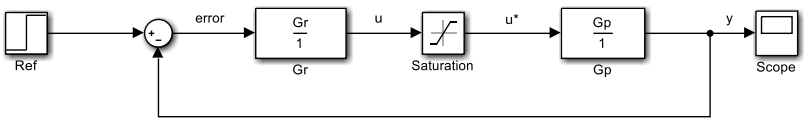


Figura 3. Diagrama de control de un proceso con saturación de la acción de control.

En un control implementado en un sistema digital la solución a esta problemática es incluir una condición en el bucle de control como esta:

```
Inicio bucle de control
    …
    if (uk <= Umax & uk >=  Umin)
    {
            Ik_1 = Ik ;
    }
    if (uk > Umax)
    {
            uk = Umax ;
    }
    if (uk < Umin)
    {
            uk = Umin ;
    }
    …
Fin del bucle
```

Existe un tercer aspecto a tener en cuenta relacionado con el muestreo de la señal de control, de error o de salida del sistema una vez diseñado e implementado nuestro controlador: las oscilaciones ocultas. Al imponer unas condiciones demasiado restrictivas al controlador, éste demandará acciones de control elevadas. Ante tales acciones, la respuesta del sistema puede variar al mismo ritmo que se realiza el muestreo, dando la sensación de que el sistema se encuentra en régimen permanente y con respuesta estable cuando en realidad el estado del sistema es todavía inestable y la respuesta oscila. Una forma de evitar esta situación es relajar las especificaciones para controladores analógicos. Sin embargo, los controladores discretos como los de tiempo finito permiten evitar este fenómeno y acotar la acción de control.

3.4. Modelado del sistema

Un modelo de un proceso nos permite diseñar un control del mismo. El modelo discreto, en nuestro caso, plantea cómo se ve el sistema objeto de estudio desde un ordenador. Tiene en cuenta que la información se recibe y se envía al sistema en instantes concretos de tiempo.

La idea que recoge este planteamiento son las ecuaciones en diferencias que describen el cambio de las señales de muestra en muestra, despreciando el comportamiento entre muestreos. La representación de señales en el dominio discreto se realiza mediante secuencias.

Nos serviremos, además, de la transformada en Z, análoga a la transformada de Laplace en sistemas continuos para facilitar la resolución y el manejo de las secuencias y las ecuaciones en diferencias [CA 2014].

Conociendo el modelo y la entrada aplicada a un sistema podremos obtener su respuesta. En [Anibal 1991] tenemos el desarrollo tras la respuesta impulsional de un sistema en el que nos basaremos a la hora de simular el modelo para obtener la salida. A grandes rasgos, la secuencia de ponderación *g(k)* de un sistema lineal invariante en el tiempo es la secuencia que se produce a la salida cuando la entrada es un impulso unitario.

En los sistemas en tiempo discreto, la secuencia de ponderación es la versión de la respuesta impulsional de los sistemas en tiempo continuo. Una vez determinada la secuencia de ponderación

que representa un sistema causal, si se aplica a partir del instante inicial *k=0*, una secuencia de entrada *{u(k)}*, la secuencia de salida puede calcularse según la suma de convolución:

$$y(k) = \sum_{j=0}^{\infty} g(k-j)u(j)$$

Ecuación 5. Salida del sistema en función de la entrada y la secuencia de ponderación

De hecho, podemos definir la función de transferencia del sistema mediante la transformada en Z de su secuencia de ponderación:

$$G(z) = Z\{g(k)\} = \sum_{k=0}^{\infty} g(k)\, z^{-k}$$

Ecuación 6. Función de transferencia del sistema en función de la secuencia de ponderación

El modelo simplificado de nuestro sistema de control estará formado por dos elementos, un controlador que reciba como entrada la señal de error y cuya salida sea la acción de control; y un proceso en el que se reciba la acción de control y produzca una salida en consecuencia. Si incluimos los convertidores A/D y D/A en el proceso tendremos que al controlador le llegará la señal ya muestreada y convertida a una secuencia digital, y que la secuencia de control que genere llegará al proceso convertida en una señal apropiada para que el sistema produzca una salida. El siguiente esquema ilustra este modelo:

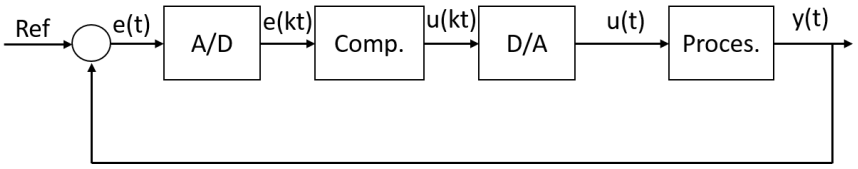


Figura 4. Diagrama de control digital en bucle cerrado.

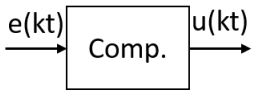


Figura 5. Controlador digital aislado con entrada y salidas como secuencias.

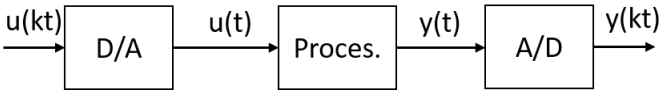


Figura 6. Señales de control y salida entre computador y proceso.

De esta manera, la salida será una función de la acción de control, ambas expresadas como secuencias:

$$\{y_k\} = f(\{u_k\})$$

Ecuación 7. Secuencia de salida en función de la secuencia de control.

Una vez introducido y comentado el concepto base del modelo del sistema discreto y las herramientas de las que nos serviremos para su manipulación vamos a abordar la obtención del

mismo.

Vamos a obtener el modelo de nuestro proceso, el motor de corriente continua con escobillas, mediante dos procedimientos. El primero, a partir de sus ecuaciones físicas, aplicando los conocimientos adquiridos en la asignatura de accionamientos electromecánicos y apoyándonos en documentación existente. El segundo planteamiento de la obtención del modelo es mediante mediciones experimentales.

Una primera posibilidad consiste en obtener una salida del sistema y compararla con las salidas que presentan los sistemas de distinto orden y obtener los parámetros de la misma mediante el modelo de estos sistemas.

La segunda posibilidad de obtención del modelo es mediante la medición experimental del comportamiento del sistema ante determinadas entradas y aproximación matemática a partir de esta respuesta.

Más adelante, en el apartado 4.3 se desarrollan las distintas opciones para optar por el mejor modelo que permita el diseño de un control óptimo.

3.5. Sensorización

Para la realizar un control en bucle cerrado necesitamos conocer la respuesta del proceso. El sensor es un elemento tan fundamental como el controlador ya que sin la realimentación de la respuesta del sistema sería imposible conseguir una señal de error que sirva de entrada a nuestro control y desconoceríamos si la acción de control calculada ha surtido el efecto esperado.

La elección del sensor concreta enormemente el desarrollo y decide en gran medida el resto de la implementación a desarrollar. Cabe destacar que el sensor no es un elemento perfecto e interfiere en el sistema, de forma que la señal proporcionada por el mismo difiere de la respuesta real, dando lugar a errores más o menos importantes en función de la calidad del sensor. Una forma de reducir el efecto de estos errores es tener en cuenta este elemento en el modelo del proceso mediante un modelado experimental como ya se ha indicado.

Existe una gran variedad de sensores para la medición de cualquier magnitud física. En el punto 4.4 se estudian distintas configuraciones de sensores y sistemas de acondicionamiento de la señal medida para decidirse por la mejor implementación para este trabajo.

3.6. Sistema de adquisición de datos

Contar con un modelo del proceso y un sensor de la salida del mismo es indispensable para implementar un control en bucle cerrado. Sin embargo, sin un dispositivo que sincronice, adquiera y procese las señales, tanto de entrada como de salida del controlador, elaborar el control sería igualmente imposible. Un sistema de adquisición de datos es el encargado de realizar estas funciones.

Un sistema de adquisición de datos es aquel que implementa el muestreo, y lleva asociado un reloj que gobierna con su señal los instantes en los que se realizan las distintas acciones. Supone, por tanto, la interfaz entre el computador y el proceso físico a controlar.

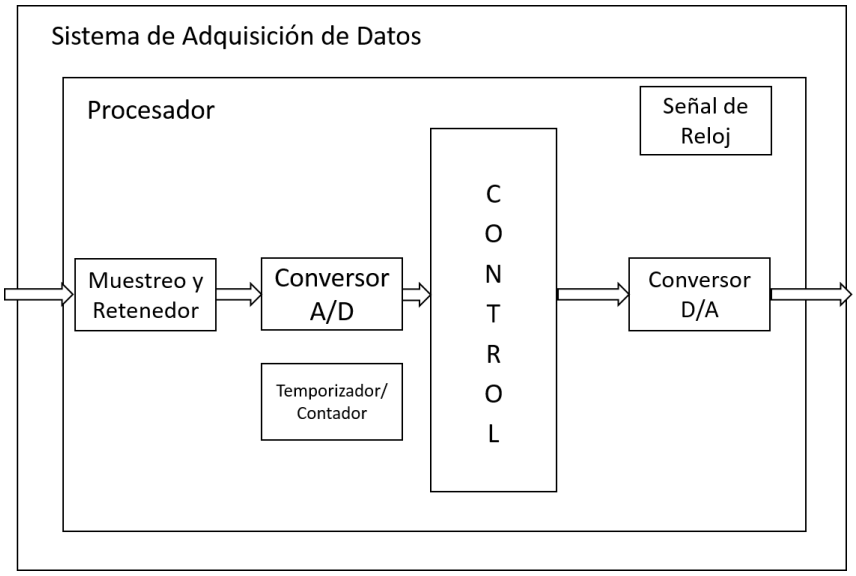


Figura 7: Diagrama de bloques de un sistema de adquisición de datos

Desde un enfoque más global, el sistema de adquisición de datos incluye el procesador en el que se implementa el controlador y todos los elementos necesarios para la adquisición, procesado y producción de señales. En el punto 4.5 se comentarán los pros y los contras de distintos sistemas de adquisición y se decidirá por uno para la realización del trabajo.

3.7. Acción de control

Ya hemos tratado el muestreo y los sensores junto al sistema de adquisición, así como nuestro proceso y su modelado. El tercer pilar necesario para poder diseñar, desarrollar e implementar un buen controlador es la acción de control que será la salida del control y la entrada del proceso, tras el acondicionamiento adecuado.

En electrónica se puede conseguir el mismo efecto de control sobre un motor mediante distintas señales de propiedades dispares. De igual forma se pueden diseñar y utilizar multitud de componentes que desarrollen estas señales a partir de características distintas: frecuencia, tensión, ciclo de trabajo, etc.

Para proteger al microcontrolador y la tarjeta de adquisición de datos es imprescindible utilizar una etapa que incorpore toda la electrónica de potencia necesaria para administrar las altas tensiones y corrientes que requiere el proceso para su funcionamiento. Esta etapa estará gobernada por las señales del controlador como entradas; la alimentación será la requerida por la carga, es decir, el proceso; y la salida será la acción de control final que llegará realmente al proceso.

Existen distintas configuraciones que permiten la realización de estas funciones, en el punto 4.6 se analizarán y se decidirá por una de ellas.

3.8. Generación de referencias

Un sistema de control necesita referencias. Cuando se trabaja con robots móviles o cualquier máquina de control numérico por computador suele ser necesario un generador de trayectorias de movimiento de manera que este sea el que indique las distintas posiciones en el espacio a las que los actuadores deben llevar el movimiento del robot en los distintos instantes de tiempo.

Uno de los posibles planteamientos es el de definir las trayectorias a partir de una serie de puntos que se pueden generar u obtener por distintos medios (sistemas de visión, etc). Este es el movimiento punto a punto. Para muchas aplicaciones, incluyendo instrumentación, sistemas de posicionamiento, medicina y automatización en general, los movimientos punto a punto son los más utilizados, por lo que su optimización tendrá el mayor impacto en el funcionamiento del sistema. De hecho, hoy en día cualquier máquina de las antes mencionadas funciona a partir de modelos en dos o tres dimensiones del trabajo a realizar. Estos modelos están parametrizados y pueden analizarse, independientemente del software del diseño, para obtener todos los puntos necesarios para su tratamiento.

Movimiento punto a punto significa que, desde el reposo, la carga es acelerada hasta una velocidad constante, tras la cual es decelerada hasta que tanto aceleración como velocidad alcanzan simultáneamente el valor 0 en el momento en el que la carga llega al destino programado.

Existen dos métodos para generar curvas a partir de un conjunto de puntos: curvas de interpolación de puntos y curvas de aproximación de puntos. El primer método produce una curva que pasa por todos y cada uno de los puntos, mientras que el segundo genera una curva que se aproxima, como su propio nombre indica, a los puntos pero puede no pasar por alguno de los puntos (o por ninguno).

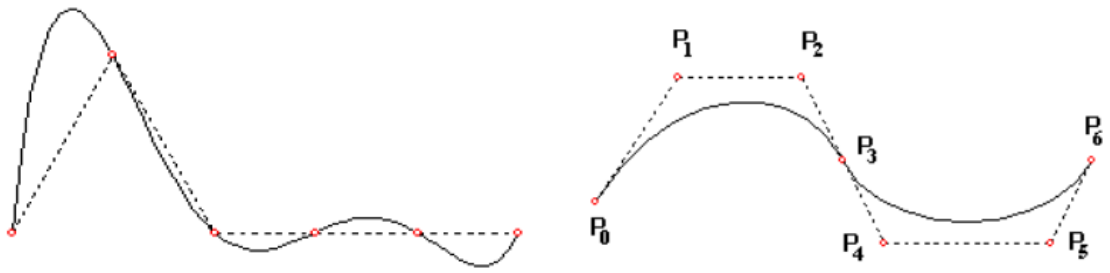


Figura 8: Interpolación (izquierda) y aproximación (derecha)

Una posible trayectoria puede basarse en la curva de Bézier de aproximación de puntos. En el caso de una interpolación de puntos podría utilizarse una Spline Cúbica Natural. Escapa al objetivo de este proyecto la realización de un generador de trayectorias, pero si se desea puede consultarse en la bibliografía [RAU 14]

Podrían invertirse horas optimizando los parámetros de los controladores de nuestro sistema y todavía no conseguir las prestaciones que deseamos. También perderíamos el tiempo si, en algún punto del proyecto o de futuras variaciones, tuviéramos que cambiar nuestros servomotores de corriente continua por motores paso a paso, que tienen una disposición física completamente distinta [CHUCK 07].

En lugar de correr el riesgo, una mejor opción es invertir el esfuerzo en el desarrollo de patrones o perfiles de movimiento, que supongan la referencia y por tanto la respuesta de nuestro sistema entre los puntos obtenidos del generador de trayectorias, que en optimizar parámetros de un

control que depende en gran medida de los ensayos y la bondad del modelo del proceso. De esta manera, habiendo obtenido un control lo más adecuado posible, pudiendo medir la salida de nuestro proceso y aplicarle una acción de control en función de esta, la última incógnita en el presente trabajo es la referencia a seguir por el sistema.

La forma definitiva de generación de trayectorias punto a punto, así como otros tipos de trayectorias, incluyendo la utilizada en las maquinas herramienta de control numérico, es construir un perfil o patrón que compense las carga especifica del motor y sus características mecánicas.

El cálculo de estos perfiles generalmente se realiza por adelantado y se guardan como variables sus parámetros. De esta forma se generan unos vectores de referencia correspondientes a la velocidad en cada punto de la curva.

La forma de concatenar movimientos consiste en generar una tabla de forma que un reloj externo controle el cambio de un elemento a otro. El sistema de control y sensor seguirán las referencias generadas por el perfil para cada elemento que se considerará como un incremento relativo (o decremento) en la posición a lo largo de un eje.

Los perfiles más usados para movimientos punto a punto son los de curva en S, concretamente su aproximación polinomial de tercer orden y su variante más simple el perfil trapezoidal. Estas y otras variaciones se tratarán en el punto 4.8.

4. Planteamiento de soluciones alternativas y justificación de la solución adoptada

A continuación se presentan las distintas soluciones planteadas para la consecución del objeto del proyecto, indicando para cada caso la opción elegida y justificando esta decisión.

4.1 Tipos de muestreo

Existen varias técnicas de implementación de un muestreo en un microcontrolador. A continuación describiremos cada una de ellas, ordenadas de mayor a menor en función del consumo de CPU, explicando al final cómo se van a implementar como parte de la solución de nuestro proyecto.

4.1.1 Muestreo por polling

Inicia el muestreo o la conversión y permanece a la espera mientras se realiza. Cuando se ha finalizado la conversión A/D tras el muestreo, o se dispone de un dato que no requiere conversión, un bit se pone a 1, esto puede ser por un pulso procedente de un oscilador o por un evento externo que genera dicho cambio, y se procede a leer el dato. Su implementación es muy simple pero se sirve por completo de la CPU y el periodo de muestreo lo determina el hardware de adquisición. El algoritmo básico en un bucle de control tiene la forma:

```
Iniciar variables
Inicio Bucle
        Iniciar la conversión de la salida del proceso
        Espera hasta fin de conversión
        Determinación de acción de control a aplicar
        Aplicar acción de control
        Actualización de variables
Fin del bucle
```

4.1.2 Muestreo por retraso software

Tras tomar la muestra, calcular la acción de control y aplicarla, se realiza una espera hasta que se completa el periodo de muestreo. Para su optimización requiere de un sistema multitarea en el que el algoritmo de control sea sólo una más de las tareas a realizar. Requerirá además de alguna función que ponga a "dormir" la ejecución hasta que se finalice el periodo, durante este tiempo, la tarea del control pasa a segundo plano y el procesador pasa a realizar otra tarea que esperaba a ejecutarse (posible aplicación para coordinación de ejes con un solo microcontrolador lo bastante rápido). Una vez cumplida la espera, se retoma la tarea principal. Para un funcionamiento que requiera exactitud en la ejecución es necesario sistemas de tiempo real que puedan manejar bien prioridades. El algoritmo básico de control sería:

Iniciar variables
Inicio Bucle
    Iniciar muestra de la salida del proceso
    Determinación de acción de control a aplicar
    Aplicar acción de control
    Actualización de variables
    Espera hasta finalizar el periodo de muestreo
Fin del bucle

4.1.3 Muestreo por interrupción

Una señal de reloj indicará cuando muestrear y guardar el dato. Esta señal interrumpe la CPU y se inicia un manejador que gestiona la interrupción con el código necesario para muestrear la señal y guardarla. Una vez acabado el muestreo se retoma la ejecución donde se había quedado. Pese a ser algo más complejo de implementar, es el más utilizado en microcontroladores. El algoritmo básico de control sería:

Iniciar variables
Rutinas principales
Flag de interrupción. Inicio del manejador
    Iniciar la conversión de la salida del proceso
    Espera hasta fin de conversión
    Determinación de acción de control a aplicar
    Aplicar acción de control
    Actualización de variables
Fin del manejador

4.1.4 Muestreo por Acceso Directo a Memoria (DMA)

El hardware de adquisición es el que manda periódicamente los datos a un buffer o memoria en el procesador o en el mismo controlador del hardware. Sigue siendo necesaria una señal de reloj, pero es el hardware el que se encarga del muestreo. En este caso no interviene el procesador en la adquisición del dato, sino que directamente se accede a la memoria cuando se necesita, pudiendo dedicar toda la potencia del procesador a tareas más interesantes y pudiendo, incluso, muestrear varias señales a la vez. El algoritmo básico de control sería:

Iniciar variables
Inicio Bucle
    Comprobación de disponibilidad de datos
    Obtención de muestra de la salida del proceso
    Determinación de acción de control a aplicar
    Aplicar acción de control
    Actualización de variables
Fin del Bucle

4.1.5 Solución adoptada

Debido a las características de nuestro tipo de microcontrolador, con una administración de interrupciones por prioridad y alta frecuencia de reloj, optaremos por el muestreo por interrupción para la lectura de datos del encoder del motor. Un manejador de la interrupción actualizará el valor de un contador al detectarse un cambio en los canales del encoder midiendo así la posición y velocidad.

Utilizaremos muestreo por DMA, gestionado por una interrupción, para la intensidad que nos dará la salida del control de par. De esta forma el uso de procesador se reducirá substancialmente siendo los propios periféricos los encargados de la administración de la toma de datos.

En cuanto al bucle de control, se optará por una sencilla implementación mediante retraso software gracias a la precisión para medir el tiempo que incorpora la placa seleccionada.

Una segunda posibilidad es una implementación por interrupción del bucle de control, de manera que un contador reduzca su valor con la frecuencia adecuada, de manera que alcance el valor 0 cada milisegundo. Cuando esto suceda, un flag activará una interrupción cuyo manejador incluye el bucle de control, de forma que se realizarán las acciones necesarias y posteriormente se realizará una espera activa hasta la activación de la siguiente interrupción.

En el punto 5 se detalla esta implementación.

4.2 Realizaciones del controlador

Como se ha indicado en el apartado de factores a tener en cuenta en la implementación del controlador, las funciones de transferencia en las que se expresa nuestro control deben expresarse en ecuaciones en diferencias que el procesador pueda manejar. Las distintas formas de implementación se muestran a continuación.

4.2.1 Forma directa

Es la forma más sencilla de implementar la función e transferencia en el controlador. Es la forma de ecuación en diferencias pura. Pese a contar con la ventaja de tener todos los valores desplazados temporalmente, lo que evita tener que recalcularlos, implica tener que almacenar en memoria n + m variables (los órdenes de denominador y numerador).

$$G_R(z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_m z^{-m}}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_n z^{-n}} = \frac{U(z)}{E(z)}$$

$$U_k = b_0 e_k + \cdots + b_m e_{k-m} - a_1 u_{k-1} - \cdots - a_n u_{k-n}$$

Ecuación 8. Controlador en Forma Directa.

4.2.2 Forma estándar

Similar a la forma directa pero requiere el uso de menos variables gracias a una sencilla transformación:

$$G_R(z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_m z^{-m}}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_n z^{-n}} = \frac{U(z)}{H(z)} \frac{H(z)}{E(z)}$$

$$\frac{U(z)}{H(z)} = b_0 + b_1 z^{-1} + \cdots + b_m z^{-m}$$

$$\frac{H(z)}{E(z)} = \frac{1}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_n z^{-n}}$$

$$U_k = b_0 h_k + \cdots + b_m h_{k-m}$$

$$h_k = \frac{1}{a_0} e_k - \frac{1}{a_0} a_1 h_{k-1} - \cdots - \frac{1}{a_0} a_n h_{k-n}$$

Ecuación 9. Controlador en Forma Estándar.

Hasta este punto la representación de la función de transferencia en ecuaciones discretas que pueda manejar el controlador en un sistema digital abarca hasta el orden n y m de la función de transferencia del controlador del que partimos, por lo que los coeficientes del denominador son los coeficientes de la implementación. La precisión en la representación de los coeficientes del controlador es finita, por lo que para órdenes elevados existirá una elevada distorsión entre los polos y los ceros del controlador respecto a los del diseño realizado.

Si suponemos, como en la ecuación general que venimos utilizando, un denominador:

$$A(z^{-1}) = 1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_n z^{-n}$$

O en z positivas:

$$A(z) = z^n + a_1 z^{n-1} + a_2 z^{n-2} + \cdots + a_n$$

Ecuación 10. Ecuación del denominador general del control.

Como se demuestra en la bibliografía [Ogata 1996], el factor que mayor distorsión produce es a_n. La sensibilidad del control a esta distorsión es mayor si los polos son múltiples. De forma que el control es muy sensible a errores computacionales si el orden del sistema es alto y los polos son cercanos a 1 (recordemos que estamos en el dominio discreto y éste es el límite de estabilidad).

Por ello la solución más efectiva es buscar representaciones como combinaciones de sistemas de primer y segundo orden como sucede con las tres formas que nos quedan.

4.2.3 Forma serie

En esta forma el controlador se expresa como producto de fracciones de primer y segundo orden, dependiendo cada función de transferencia del producto de los polos de la función de transferencia original. Esta es la mejor opción en el caso de tener polos múltiples.

$$G_R(z) = \frac{U(z)}{E(z)} = \frac{b_0 + b_1 z^{-1} + \cdots + b_m z^{-m}}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_n z^{-n}} = G_1(z) G_2(z) \ldots G_l(z)$$

Ecuación 11. Controlador en Forma Serie

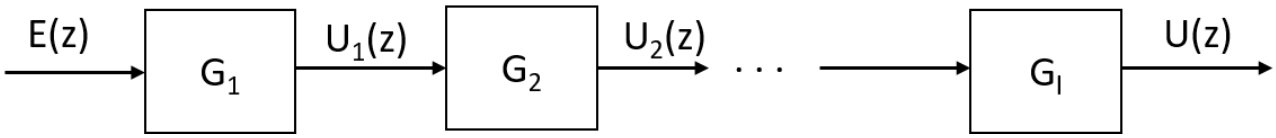


Figura 15. Diagrama de la forma serie.

4.2.4 Forma paralela

En esta ocasión, el controlador es el resultado de la suma de fracciones de primer y segundo orden.

$$G_R(z) = \frac{U(z)}{E(z)} = \frac{b_0 + b_1 z^{-1} + \cdots + b_m z^{-m}}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_n z^{-n}} = A + G_1(z) + G_2(z) + \cdots + G_q(z)$$

Ecuación 12. Controlador en Forma Paralela.

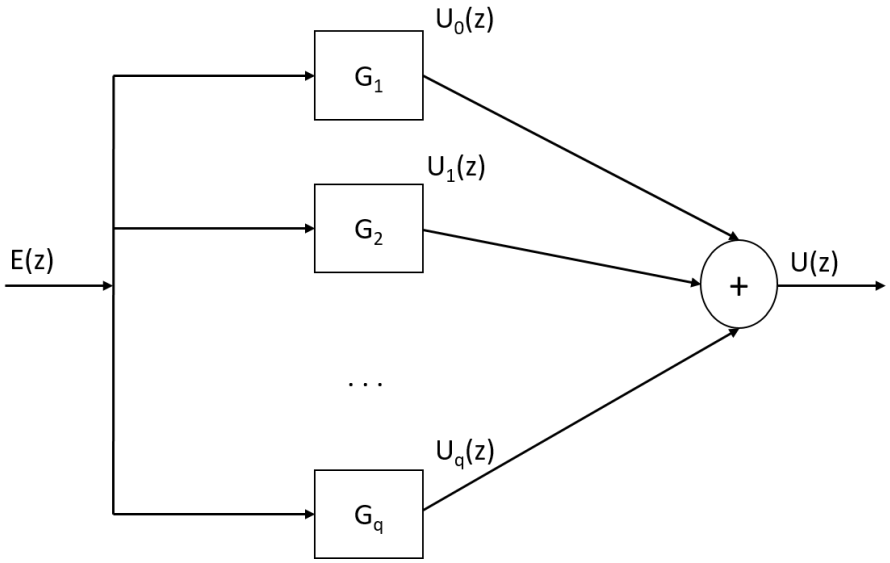


Figura 16. Diagrama de la forma paralela.

4.2.5 Forma de escalera

El controlador se expresa como una expansión continúa de fracciones siendo:

$$grado\ \big(U(z)\big) = grado\big(E(z)\big) = n$$

$$G_R(z) = \frac{U(z)}{E(z)} = \frac{b_0 + b_1 z^{-1} + \cdots + b_m z^{-m}}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_n z^{-n}} = \alpha_n + G_1^{(B)}(z)$$

$$G_i^{(B)}(z) = \frac{1}{\beta_i\, z + G_i^{(A)}(z)} \quad para\ i = 1,2,\dots,n-1$$
$$G_i^{(A)}(z) = \frac{1}{\alpha_i\, z + G_{i+1}^{(B)}(z)} \quad para\ i = 1,2,\dots,n-1$$

Ecuación 13. Controlador en forma de escalera.

Pudiendo determinar α_i y β_i como:

$$\alpha_0 = B(z)\ div\ A(z)$$
$$A_1(z) = A(z)$$
$$B_1(z) = B(z)\ mod\ A(z)$$

E iterando desde i=1 hasta n:

$$\beta_i = (A_i)\ div\ (zB_i)$$
$$\alpha_i = B_i\ div\ A_{i+1}$$
$$A_{i+1} = (A_i)\ mod\ (zB_i)$$
$$B_{i+1} = (B_i)\ mod\ (A_{i+1})$$

Ecuación 14. Obtención de los parámetros de la forma de escalera.

Pudiendo expresar la representación mediante las siguientes funciones de estado:

$$\beta_1 x_1(k+1) = \frac{1}{\alpha_1}\big(x_2(k) - x_1(k)\big) + e(k)$$
$$\beta_2 x_2(k+1) = \frac{1}{\alpha_1}\big(x_2(k) - x_1(k)\big) + \frac{1}{\alpha_2}\big(x_3(k) - x_2(k)\big)$$
$$\dots$$
$$\beta_i x_i(k+1) = \frac{1}{\alpha_{i-1}}\big(x_{i-1}(k) - x_i(k)\big) + \frac{1}{\alpha_i}\big(x_{i+1}(k) - x_i(k)\big)$$
$$\dots$$
$$\beta_n x_n(k+1) = \frac{1}{\alpha_{n-1}}\big(x_{n-1}(k) - x_n(k)\big) - \frac{1}{\alpha_n} x_n(k)$$
$$U_k = x_1(k) + \alpha_0 e(k)$$

Ecuación 15. Funciones de estado de la forma de escalera.

De esta forma podemos expresar el regulador discreto en formas implementables en el microcontrolador.

La decisión de utilizar una forma u otra dependerá enormemente del control en cuestión a

desarrollar, por lo que no nos pronunciaremos sobre este punto hasta la implementación.

4.3 Modelos del proceso

Como ya se ha explicado, es necesaria la obtención de un modelo para el diseño de un control. Vamos a proceder a la obtención el nuestro por tres métodos distintos y decidiremos cuál utilizaremos para el control de nuestro proyecto.

4.3.1 Modelado matemático

El motor de corriente continua es una máquina eléctrica constituida por dos elementos principales [Acc. Elec. 15]:

- Un circuito magnético, formado por dos armaduras y el espacio comprendido entre ellas denominado entrehierro. La armadura fija, llamada *inductor* o estator, es de material magnético macizo y está compuesta:

- Por dos polos, con un núcleo cada uno, sobre los cuales está bobinado un devanado inductor, destinado a generar el campo magnético inductor, la inducción es magnética radial.

- Por unas piezas polares que contribuyen a distribuir la inducción magnética sobre una parte mayor de la periferia del inducido. El estator, además, tiene dos culatas destinadas a canalizar el flujo inductor.

La otra armadura es llamada rotor o *inducido* y está constituida, normalmente, por un apilamiento de chapas ferromagnéticas, aisladas entre ellas y dispuestas perpendicularmente al eje de rotación. Las dos armaduras estator y rotor, están separadas por el entrehierro. En la periferia del inducido se colocan en ranuras los conductores del devanado inducido, paralelos al eje de rotación.

Los conductores están conectados entre sí de forma que constituyen un devanado cerrado sobre sí mismo.

- Un colector autoconmutador mecánico, que tiene como misión “conducir” la corriente que se inyecta, a través de las escobillas, a los conductores del devanado inducido (si trabaja como motor). También tiene la misión o función de conmutar la corriente y además realiza la función de “sensor de posición”. Tiene, también, la función de convertir corrientes continuas en alternas (ondulador) y viceversa (rectificador) cuando trabaja como motor de c.c. o generador de c.c. respectivamente.

Magnéticamente, un colector transforma para un observador exterior fijo, un campo magnético giratorio en un campo magnético fijo espacialmente y temporalmente.

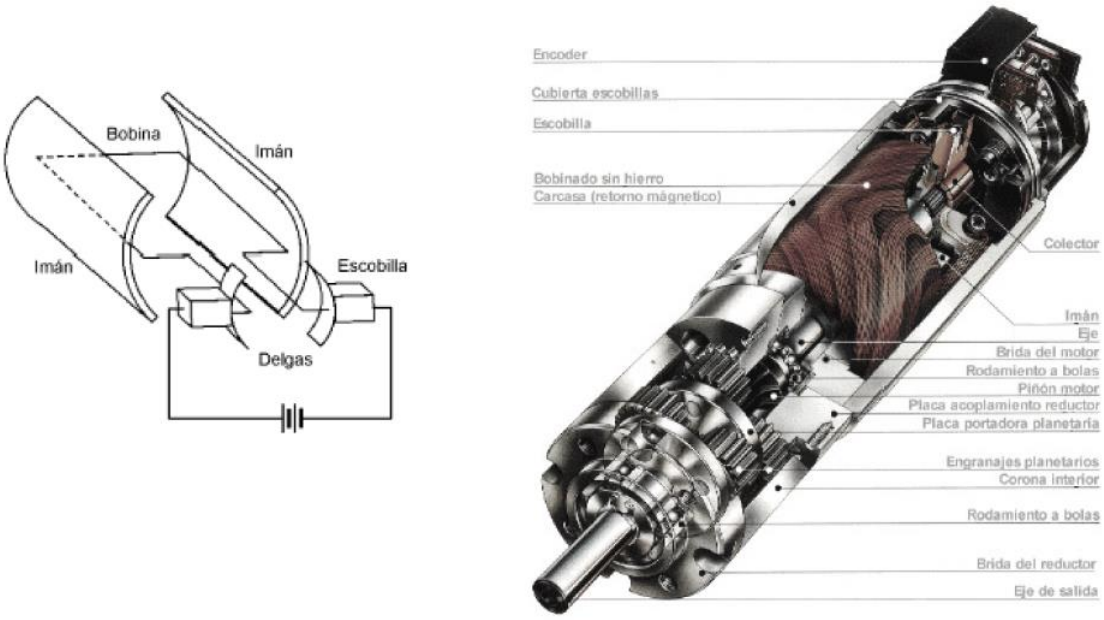


Figura 9. Esquema de las partes de un motor de corriente continúa con escobillas [Acc. Elec11].

El modelo matemático del motor de corriente continua es el que sigue. Si bien presentaremos solo las principales ecuaciones que describen el modelo matemático del motor, el desarrollo en profundidad estudiando la generación y efectos del campo magnético, del cual proceden estas principales ecuaciones, puede verse en la bibliografía [Acc. Elec11] [Acc. Elec15] [Serrano 1989].

Las dos magnitudes más interesantes para el estudio del funcionamiento del motor son el par y la velocidad.

En el inducido:

$$U(t) = E_c(t) + R \cdot I(t) \quad (1)$$

Siendo:

$$E_c(t) = \frac{n(t)}{60} N \cdot \hat{\phi}_c(t) \quad (2)$$

Además el par será:

$$T(t) = \frac{1}{2\pi}\frac{n}{60}\hat{\phi}_c(t) \cdot I(t) \quad (3)$$

Por lo que la expresión de la velocidad será:

$$n(t) = \frac{(U(t) - R \cdot I(t))}{N \cdot \hat{\phi}_c(t)} 60 \approx \frac{U(t)}{N \cdot \hat{\phi}_c(t)} 60 \quad (Ref1)$$

Ecuación 16. Expresión de la velocidad a partir de las expresiones de tensión (1), fem (2) y par (3).

El balance de energía, despreciando las pérdidas en el entrehierro y las pérdidas mecánicas, suponiendo que la potencia aportada al motor es: P_{abs}

$$P_{abs} = P_{J1} + P_{ind} + (E \cdot I)$$

Donde:

$$P_{J1} = R_1 \cdot {I_1}^2$$

$$P_{ind} = R_i \cdot I^2$$

$$E(t) \cdot I(t) = T(t) \cdot \Omega(t) = \frac{T(t) \cdot 2 \cdot \pi \cdot n(t)}{60}$$

Ecuación 17. Ecuación del balance de energía

La curva par-velocidad, muy común en los motores eléctricos, T=f(Ω) es un índice de la estabilidad del motor.

Las cargas, a su vez, también se caracterizan por sus respectivas curvas par-velocidad pero, en este caso, se trata de un par resistente.

La intersección de estas dos curvas, la del par resistente y la del par motor, definen el punto de funcionamiento en régimen permanente y éste puede ser un punto de equilibrio estable o no. Además dicha intersección fija el valor del parámetro corriente inducida.

Para estudiar la condición de estabilidad, haremos uso de la ecuación fundamental de la dinámica:

$$J\frac{d\Omega}{dt} = T_m - T_R$$

Ecuación 18. Ecuación fundamental de la dinámica.

Siendo:

T_m ⇒ Par motor

T_R ⇒ Par resistente

J ⇒ Conjunto de inercias de las partes rotatorias

Ω ⇒ Velocidad del motor

En el caso de la curva:

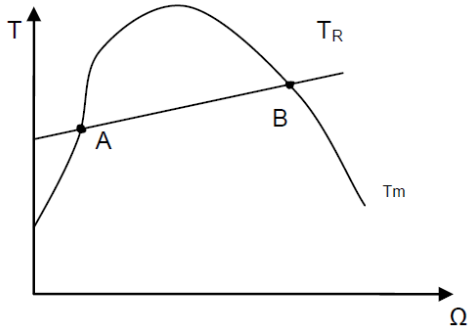


Figura 10. Curva par velocidad característica e intersección con el par resistente.

Los dos puntos A y B se corresponden con dos posibles puntos de funcionamiento.

Supongamos que funcionamos en el punto B y que momentáneamente se ralentiza el motor, produciendo, como consecuencia, un aumento instantáneo del par.

Si la velocidad disminuye el par motor aumenta y $(T_m-T_R)>0$ con lo que se acelera el motor y vuelve al punto B, y ello tanto más rápidamente como distantes estén las curvas T_m y T_R.

Si por el contrario, la velocidad aumentara, $T_R>T_m$ y entonces el motor decelera para volver al punto B.

Si se hacen los mismos razonamientos en el punto A, observamos que si el motor decelera entonces el par resistente $T_R>T_m \Rightarrow$decelera más el motor y se para. Si en vez de decelerar, el motor acelera momentáneamente entonces dejará el punto A para ir a estabilizarse en B, único punto estable de funcionamiento.

Concluimos diciendo que la condición de estabilidad es que la curva del par motor tenga menos pendiente que la del resistente, o matemáticamente:

$$\frac{\partial T_m}{\partial \Omega} < \frac{\partial T_r}{\partial \Omega}$$

Al final, las ecuaciones que definen el funcionamiento del motor de corriente continua son

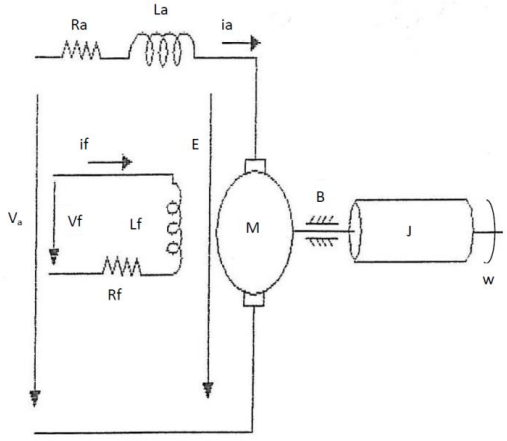


Figura 11. Modelo matemático del motor CC [CA 2014].

$$T_m = K_T \cdot i_a$$

$$E = K_v \cdot i_a$$

$$V_a = R_a \cdot i_a + L_a \frac{di_a}{dt} + e$$

$$T_m = J \frac{di_a}{dt} + B \cdot w + T_r$$

$$P_m = T_m \cdot w$$

$$P_i = \frac{1}{T} \int i_a \cdot v_a \cdot dt$$

Ecuación 19. Formulación del modelo matemático del motor CC.

Donde:

V_a⇒Tensión media en la armadura

R_a⇒ Resistencia de armadura

L_a⇒ Inductancia de armadura

I_a⇒ Corriente media que atraviesa la armadura.

E⇒ Fuerza electromotriz

w⇒ Velocidad angular del eje

T_m⇒ Par motor

T_r⇒ Par resistente

K_v y K_T⇒ Constantes del motor

B⇒ Coeficiente de rozamiento

J⇒ Inercia del motor

P_m⇒ Potencia mecánica

P_i⇒ Potencia eléctrica

A efectos de notación, se puede definir (*Ra, La)* como la resistencia y la inductancia de la armadura, *E* como la fuerza electromotriz $E = k_1 i_f \omega$, donde ω es la velocidad del eje. En la parte mecánica, el par motor $T = k_2 i_f i_a$ se equilibra por un componente inercial $J_{\dot{\omega}}$, una componente de fricción $B\omega$ y un par de carga aplicado T_L, el cual puede cambiar con el tiempo pero se asume que

es independiente de ω. Por consiguiente, se puede distinguir dos ecuaciones de funcionamiento:

$$\text{Eléctrica: } V_a = R_a i_a + L_a \frac{di_a}{dt}\omega + k_1 i_f \omega$$

Ecuación 20 Ecuación eléctrica del modelo matemático del motor CC.

$$\text{Mecánica: } T = k_2 i_f i_a = J_{\dot{\omega}} + B\omega + T_L \ (Ref2)$$

Ecuación 21. Ecuación mecánica del modelo matemático del motor CC.

Nótese que el sistema será lineal si i_f es solo una constante (control por armadura). Asumiendo $k_i i_f = K_{fi}$ y un esquema en bucle cerrado, no es complicado llegar a la función de transferencia entre la tensión en armadura (causa) y el giro del eje (efecto).

$$V_a(s) - K_{f1}\omega(\mathrm{s}) = R_a I_a(\mathrm{s}) + L_a s I_a(\mathrm{s})$$

$$\frac{I_a(\mathrm{s})}{V_a(s) - K_{f1}\omega(\mathrm{s})} = \frac{1}{R_a + sL_a}$$

$$K_{f2} I_a(s) - T_L(s) = Js\omega(\mathrm{s}) + B\omega(\mathrm{s})$$

$$\frac{\omega(\mathrm{s})}{K_{f2} I_a(s) - T_L(s)} = \frac{1}{Js + B}$$

$$\frac{\omega(\mathrm{s})}{V_a(s)} = \frac{K_{f2}}{(R_a + sL_a)(Js + B) + K_{f1}K_{f2}}$$

Ecuación 22. Función de transferencia del modelo de velocidad del motor CC.

Habitualmente el valor de la inductancia L_a es despreciable por lo que se suele decir que se tiene un *sistema de primer orden* y por tanto de comportamiento sobreamortiguado. Además para la mayoría de los motores, los valores de $K_{f1}\ y\ K_{f2}$ suelen ser tan aproximados que se pueden tomar como el mismo valor reduciéndose a una sola constante multiplicada por sí misma.

Sustituyendo con los valores proporcionados por el fabricante en la hoja de características de nuestro modelo [Anexo 3.1], sabiendo que trabajaremos dentro de la zona segura indicada por el fabricante, limitando el par a 0.028 Nm, la velocidad a 5000 rpm, y la intensidad a 2 A, nos sitúa en la tabla 2 en 9.55V (con velocidad máxima entre 3570 y 5000 rpm).

Teniendo en cuenta que el parámetro de fricción se da en unidades Nm s/rad, es decir por vuelta, debemos multiplicar este valor por el número de vueltas máximo de nuestro punto de funcionamiento antes descrito:

$$2.3\ 10^{-6} Nm \frac{s}{rad}\frac{2\pi\ rad}{rev}\frac{min}{60s}\frac{3570 rev}{min} = 8.59\ 10^{-4}\ Nm$$

Se tiene:

$$\frac{\omega(\mathrm{s})}{V_a(s)} = \frac{0.0188}{\left((0.84\ 10^{-3})s + 1.08\right)\left((3.2\ 10^{-6})s + 8.59\ 10^{-4}\right) + 0.0188^2}$$

Simplificando:

$$\frac{\omega(s)}{V_a(s)} = \frac{6994047.619}{(s + 1285.714286)(s + 266.875) + 131488.1}$$

$$\frac{\omega(s)}{V_a(s)} = \frac{6994047.619}{s^2 + 1552.58s + 474613.02}$$

Ecuación 23. Función de transferencia del modelo de velocidad de nuestro motor CC.

Con polos en:

$$s = -418.5 \; y \; s = -1134.079$$

$$\frac{\omega(s)}{V_a(s)} = \frac{6994047.619}{(s + 418.5)(s + 1134.079)}$$

Sería un sistema estable y cuya ganancia en bucle abierto (s=0) sería de 14.7.

Si bien el modelo obtenido parece razonable difiere bastante de la realidad. Pues hay que destacar la presencia de una zona muerta en el motor en los valores de tensiones en torno a 0 debido a las inercias y rozamientos que el eje del motor debe vencer para comenzar el movimiento. Esto obliga a tener en cuenta que en la obtención del modelo de forma experimental debemos partir del movimiento a la hora de realizar el ensayo.

Podemos obtener de la misma forma la posición del eje en función de la tensión ya que la velocidad del eje es la derivada de la posición de este. Luego si introducimos un integrador en el sistema:

$$\frac{\theta(s)}{V_a(s)} = \frac{6994047.619}{(s + 418.5)(s + 1134.079)s}$$

Ecuación 24. Función de transferencia del modelo de posición de nuestro motor CC.

El par del motor es función de la intensidad, y esta a su vez es función de la tensión y de la f.e.m. que depende a su vez de la velocidad. Por tanto una posibilidad para controlar el par es controlar tensión y considerar la velocidad como una perturbación en la tensión. Que podemos prealimentar y eliminar ya que la conocemos y nuestro control es suficientemente rápido [CA 2014].

$$V_a(s) - K_{f1}\omega(s) = (R_a + L_a s)\, I_a(s)$$

$$I_a(s) = \frac{V_a(s) - K_{f1}\omega(s)}{(R_a + L_a s)}$$

$$\frac{I_a(s)}{V_a(s) - K_{f1}\omega(s)} = \frac{1}{R_a + sL_a}$$

$$T(s) = K_{f2} I_a(s)$$

$$\frac{T(s)}{I_a(s)} = K_{f2}$$

Ecuación 25. Control de par para el motor CC.

En cuanto al modo de funcionamiento del motor, dado que la fuente de alimentación del devanado inductor del motor es externa e independiente, hablaremos de un motor de excitación independiente. Si suponemos que el motor trabaja en régimen nominal, podemos deducir las curvas características:

$$n = f(I)\ ;\ \ T = f(I)\ ;\ \ T = f(n)$$

De la ecuación eléctrica de tensiones (Ref 1) se deduce que la curva n = f (I) es una recta de pendiente negativa pero reducida ya que (RI) es reducido frente a U_N.

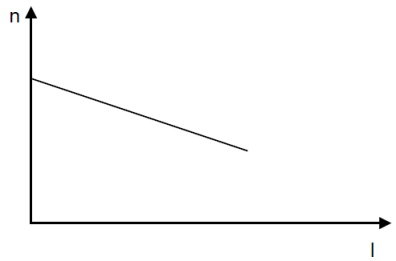


Figura 12. Diagrama de control digital en bucle cerrado.

De la ecuación del par (Ref 2) para n = cte., T = f (I) es una recta que pasa por el origen:

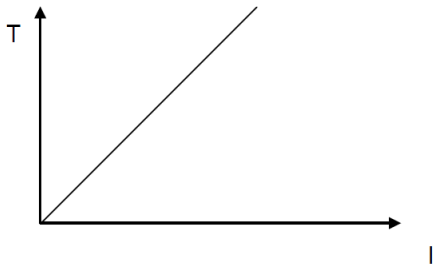


Figura 13. Diagrama de control digital en bucle cerrado.

De la ecuación eléctrica y de la ecuación del par se obtiene:

$$T = \frac{1}{2\pi} n \cdot \hat{\phi}\left(U_n - n \cdot N \cdot \hat{\phi}_n\right) = f(n)$$

Ecuación 26. Curva Par-Velocidad.

Es una recta de pendiente negativa, de modulo elevado ya que R es pequeña.

De esta forma tenemos los modelos de posición, velocidad y par del motor de corriente continua con escobillas a partir de las ecuaciones que describen su funcionamiento. Sin embargo, estos modelos son inexactos, como ya se ha indicado y no solo porque no incluyan las discontinuidades, si no porque no incluyen, al contrario que con los modelos experimentales, las variaciones que introducen el sensor, la reductora y la etapa de potencia en el modelo.

Los siguientes dos métodos de modelado se obtienen a partir de la respuesta del sistema ante distintas entradas. Incluyen por tanto en el modelo obtenido todos los elementos que afectan al sistema entre la entrada y la salida. Estos procesos de modelado se componen de cuatro etapas:

1.- Planificación experimental: introduciremos una entrada determinada al sistema para leer datos que indiquen el comportamiento del mismo.

2.- Selección de la estructura del modelo (primer o segundo orden en función de si hablamos de velocidad o posición).

3.- Estimación de los parámetros del modelo.

4.- Validación

4.3.2 Modelado no paramétrico. Respuesta ante escalón

Cuando necesitamos trabajar con sistemas físicos hace falta obtener las ecuaciones diferenciales que rigen el comportamiento de ese sistema. Sin embargo, estas ecuaciones resultan incómodas para trabajar, por lo que se recurre a una herramienta matemática para simplificar el trabajo, la transformada de Laplace.

Los sistemas más simples que podemos encontrar son sistemas de primer orden, que responden a una ecuación diferencial de primer grado:

$$\dot{y} + ay = bu$$

Donde *u* es la entrada, *y* la salida, y *a*, *b* son constantes del sistema.

Aplicando la transformada de Laplace:

$$sy(s) + ay(s) = bu(s)$$

Y agrupando términos:

$$y(s)[s + a] = bu(s)$$

Si obtenemos el cociente entre y(s) y u(s):

$$G(s) = \frac{y(s)}{u(s)} = \frac{b}{s + a}$$

Siendo este cociente la función de transferencia (fdt) que indica la relación entre la salida y la entrada. Si dividimos por el término *a* obtendremos la expresión normalizada:

$$G(s) = \frac{y(s)}{u(s)} = \frac{b}{s + a} = \frac{k}{1 + \tau s}$$

Ecuación 27. Función de transferencia de un sistema de primer orden.

Donde *k* es la ganancia estática y τ la constante de tiempo del sistema. Como su nombre indica, la ganancia estática estará relacionada con el valor de la respuesta cuando se estabilice, y la

constante de tiempo con el que le lleva alcanzar este valor estable.

Hasta aquí tenemos la respuesta del sistema ante una entrada en bucle abierto, lo que podemos llamar la función de transferencia del proceso. La relación en bucle cerrado del sistema sería:

$$G_{BC}(s) = \frac{G_p(s)}{1 + G_p(s)}$$

Siendo el denominador igualado a 0 lo que se denomina la ecuación característica del sistema y la que da la respuesta dinámica y estática del sistema.

$$1 + G_p(s) = 0$$

Podemos, mediante esta expresión, obtener la ganancia y la constante de tiempo en bucle cerrado:

$$k_{BC} = \frac{k_{BA}}{1 + k_{BA}}$$

$$\tau_{BC} = \frac{\tau_{BA}}{1 + k_{BA}}$$

Se puede asociar el proceso de un motor de corriente continua que relaciona la tensión de entrada con la velocidad con una fdt. de primer orden:

$$G_M(s) = \frac{\dot{\theta}}{v} = \frac{k}{1 + \tau s}$$

Partiendo de la fdt. de un sistema de primer orden, para ver el comportamiento del sistema ante una entrada de escalón unitario:

$$Y(s) = \frac{k}{1 + \tau s} \cdot \frac{1}{s}$$

Si utilizamos la transformada inversa de Laplace:

$$y(t) = K\left(1 - e^{\frac{-t}{\tau}}\right)$$

Se corresponde a una función exponencial. Para obtener K se dividirá el valor final de la salida entre el valor de la entrada, que en este caso, al ser unitaria, será 1. De esta forma el valor de la ganancia será directamente el de la salida del proceso. Para τ, si suponemos que el tiempo *t* es igual a la constante de tiempo:

$$y(\tau) = K\left(1 - e^{\frac{-t}{\tau}}\right) = K(1 - e^{-1}) = 0.632K$$

Por lo que el valor de τ será el tiempo en el que la salida alcanza el 63.2 % del valor final desde que aparece el escalón.

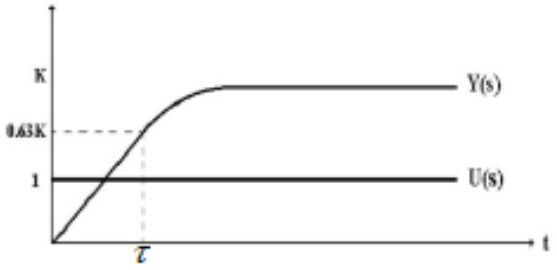


Figura 14. Identificación mediante respuesta ante escalón para sistema de tipo 0.

Si en vez de introducirse un escalón unitario se introdujera un escalón de valor *d*, siendo el valor final de la salida del proceso *c*, la ganancia del proceso sería según se ha indicado:

$$k = \frac{c}{d}$$

Este método de identificación puede no resultar apropiado para procesos complejos.

Para un sistema de tipo 1 (un integrador en la cadena directa) la respuesta ante escalón es una rampa. Esto ocurre en el caso de tener como sistema un motor cuya salida sea la posición de su eje. La tensión aplicada como acción de control, si es constante, genera un par que mantiene una velocidad constante, por lo que la posición crece de forma lineal. Podemos por tanto, aproximar este sistema mediante la integración de la velocidad de giro del eje.

La expresión general de un sistema de tipo 1 es:

$$G(s) = \frac{y(s)}{u(s)} = \frac{k}{s(1+\tau s)}$$

Siendo de nuevo los parámetros a identificar k y τ. La ganancia se puede asociar a la pendiente *m* de la respuesta del sistema. Una vez más, en el caso de tener como entrada un escalón unitario, k coincidirá con *m*. Pero para el caso de un escalón no unitario, la ganancia coincidirá con el cociente entre la pendiente y el valor del escalón (*d*):

$$k = \frac{m}{d}$$

Para obtener la constante de tiempo, ha de apreciarse el retardo entre la respuesta del sistema y una recta de ordenada 0 para el instante de inicio del escalón y pendiente *m*.

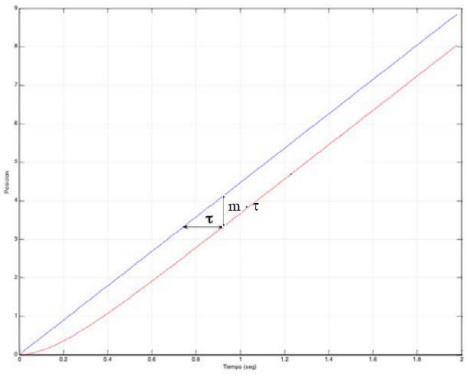


Figura 15. Identificación mediante respuesta ante escalón para sistema de tipo 1.

Presentado el desarrollo teórico de este proceso de modelado, se procede a la obtención práctica del modelo del sistema mediante la identificación de parámetros antes descrita. Esta se llevará acabo utilizando el microcontrolador como tarjeta de adquisición de datos mediante un sencillo código que aplicará una entrada de escalón y guardará lecturas del encoder en un archivo de texto. El código inicializa dos timers, uno para leer el encoder y otro para utilizar una salida PWM.

La media en pulsos se pasaría a radianes mediante la relación:

$$\frac{360\ pulsos}{1\ vuelta}\frac{1\ vuelta}{2\pi\ rad} \rightarrow 1\ pulso = \frac{2\pi}{360} rad$$

Ecuación 28. Relación pulsos-radianes de la medida del encoder.

Utilizaremos como periodo de muestreo el que ya se ha calculado anteriormente: 10ms.

El código utilizado se incluye en los anexos [Anexo A1.1], junto con la debida explicación de cada función en los comentarios del mismo. También se incluyen los *scripts* de Matlab utilizados.

Si se guardan los datos en un fichero de texto y con Matlab hacemos una gráfica de la respuesta obtenida podemos identificar los parámetros de la función de transferencia como se muestra a continuación.

Para la posición del motor con una entrada de escalón de un ciclo de trabajo de 25%:

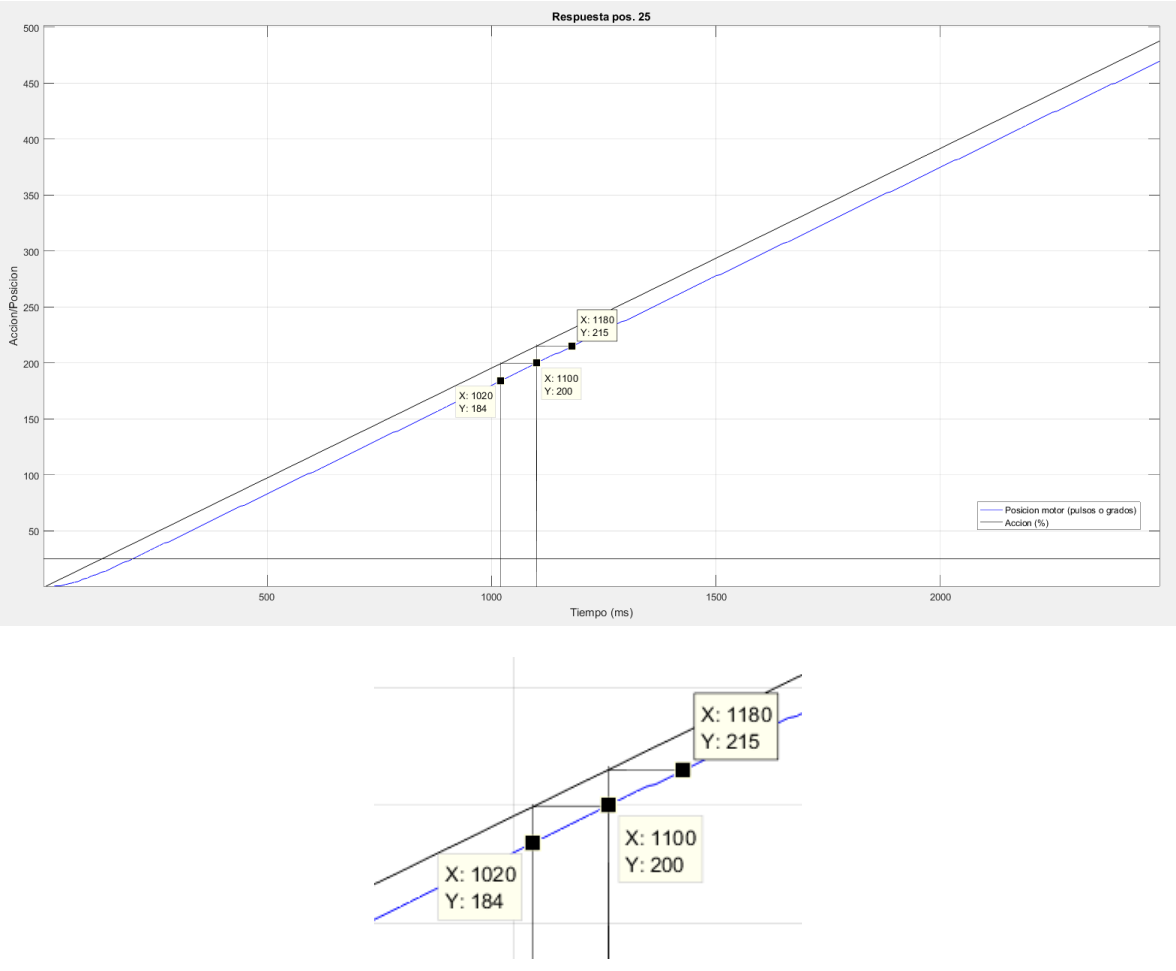


Figura 16. Respuesta ante una entrada del 25%.

Donde:

$$\tau = 1100 - 1020 = 80\,ms = 0.08\,s$$

$$m \cdot \tau = 215 - 200 = 15$$

$$m = \frac{15}{0.08} = 187.5$$

Siendo el incremento del escalón de entrada d:

$$d = 25$$

La ganancia del modelo será:

$$k = \frac{m}{d} = \frac{187.5}{25} = 7.5$$

Y el modelo queda:

$$G(s) = \frac{7.5}{s(1 + 0.08 \cdot s)}$$

Para la posición del motor con una entrada de escalón de un ciclo de trabajo de 50%:

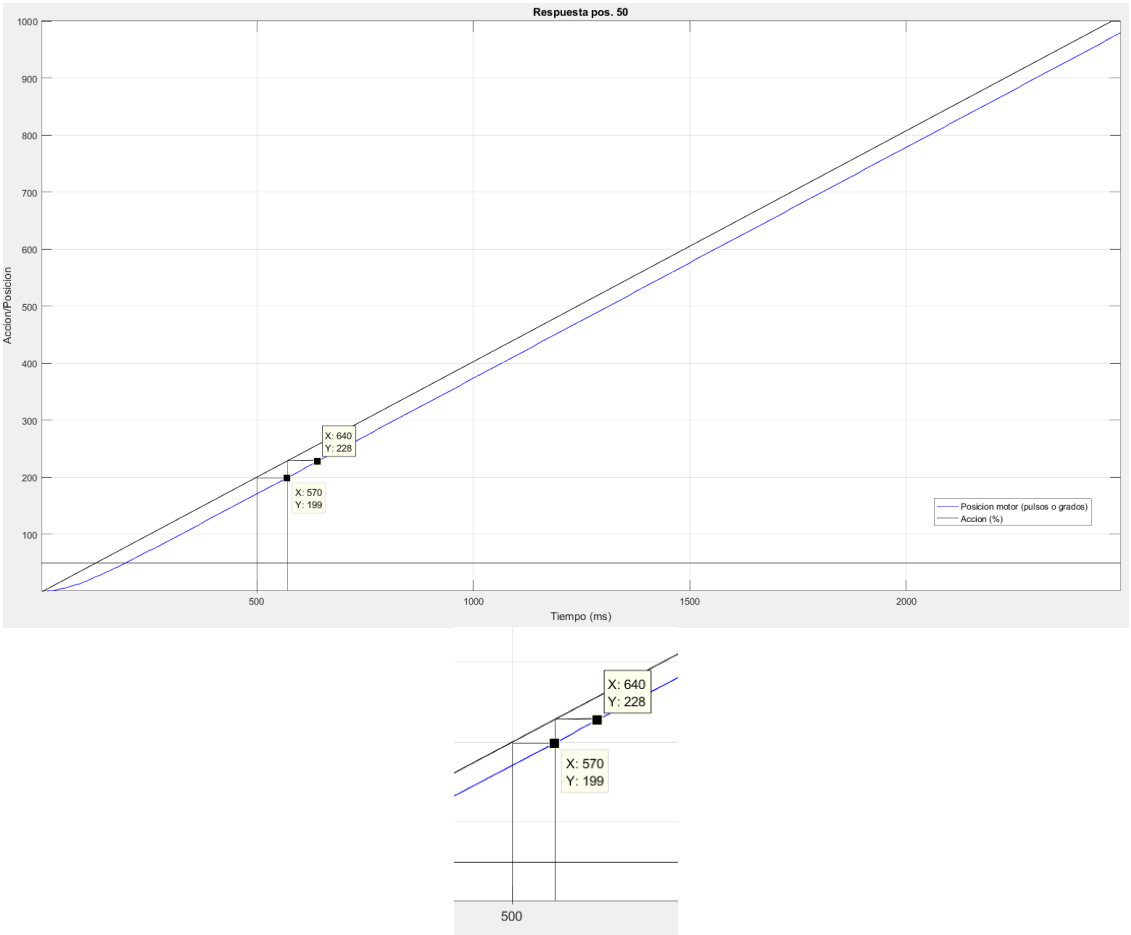


Figura 17. Respuesta ante una entrada del 50%.

Donde:

$$\tau = 570 - 500 = 70\, ms = 0.07\, s$$

$$m \cdot \tau = 228 - 199 = 29$$

$$m = \frac{37}{0.07} = 414.28$$

Siendo el incremento del escalón de entrada d:

$$d = 50$$

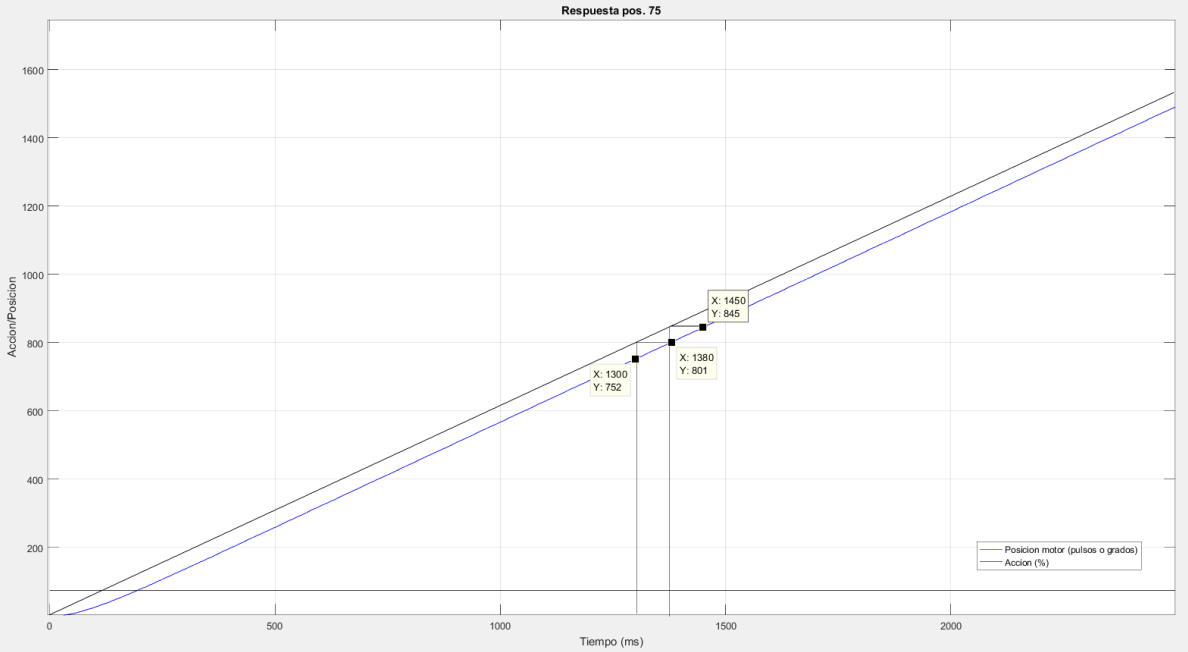
La ganancia del modelo será:

$$k = \frac{m}{d} = \frac{414.28}{50} = 8.28$$

Y el modelo queda:

$$G(s) = \frac{8.28}{s(1 + 0.07 \cdot s)}$$

Para la posición del motor con una entrada de escalón de un ciclo de trabajo de 75%:



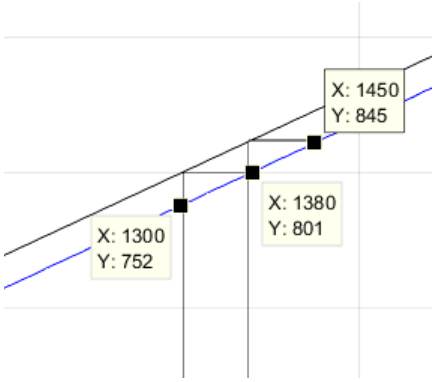


Figura 18. Respuesta ante una entrada del 75%.

Donde:

$$\tau = 1380 - 1300 = 80\,ms = 0.08\,s$$

$$m \cdot \tau = 845 - 801 = 44$$

$$m = \frac{55}{0.08} = 550$$

Siendo el incremento del escalón de entrada d:

$$d = 75$$

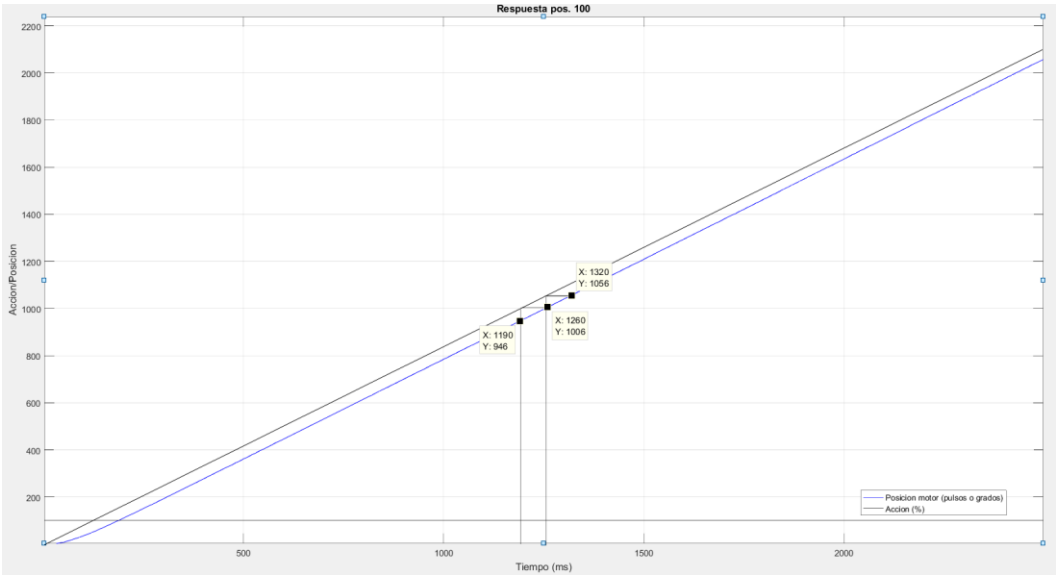
La ganancia del modelo será:

$$k = \frac{m}{d} = \frac{550}{75} = 7.33$$

Y el modelo queda:

$$G(s) = \frac{7.33}{s(1 + 0.08 \cdot s)}$$

Para la posición del motor con una entrada de escalón de un ciclo de trabajo de 100%:



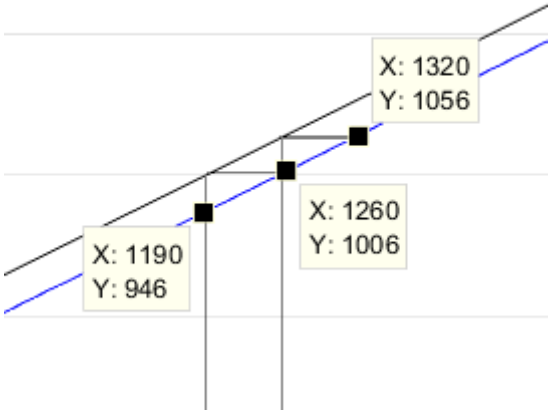


Figura 19. Respuesta ante una entrada del 100%.

Donde:

$$\tau = 1260 - 1190 = 70\ ms = 0.07\ s$$

$$m \cdot \tau = 1056 - 1006 = 50$$

$$m = \frac{59}{0.07} = 714.29$$

Siendo el incremento del escalón de entrada d:

$$d = 100$$

La ganancia del modelo será:

$$k = \frac{m}{d} = \frac{714.29}{100} = 7.14$$

Y el modelo queda:

$$G(s) = \frac{7.14}{s(1 + 0.07 \cdot s)}$$

Una vez obtenidos los 4 modelos para las distintas medidas ante variaciones en la entrada también distintas, promediamos para obtener un modelo de la posición del motor frente a la entrada:

$$G(s) = \frac{7.56}{s(1 + 0.075 \cdot s)}$$

Y mediante el siguiente esquema de Simulink validar gráficamente el modelo.

Con el siguiente esquema de Simulink vamos a validar el modelo, comprobando la respuesta ante las mismas entradas y comparando con la respuesta experimental:

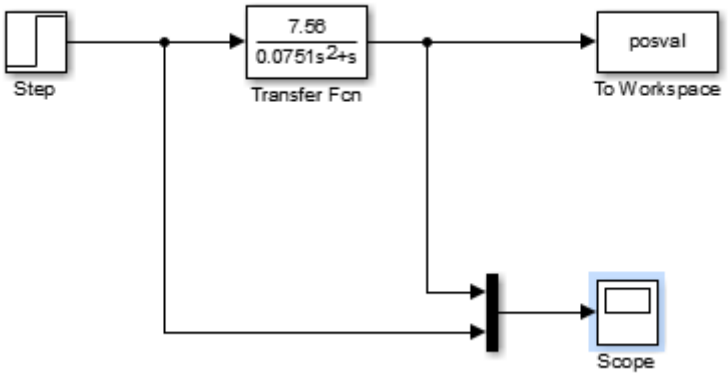


Figura 20. Esquema de simulación para comprobación del modelo

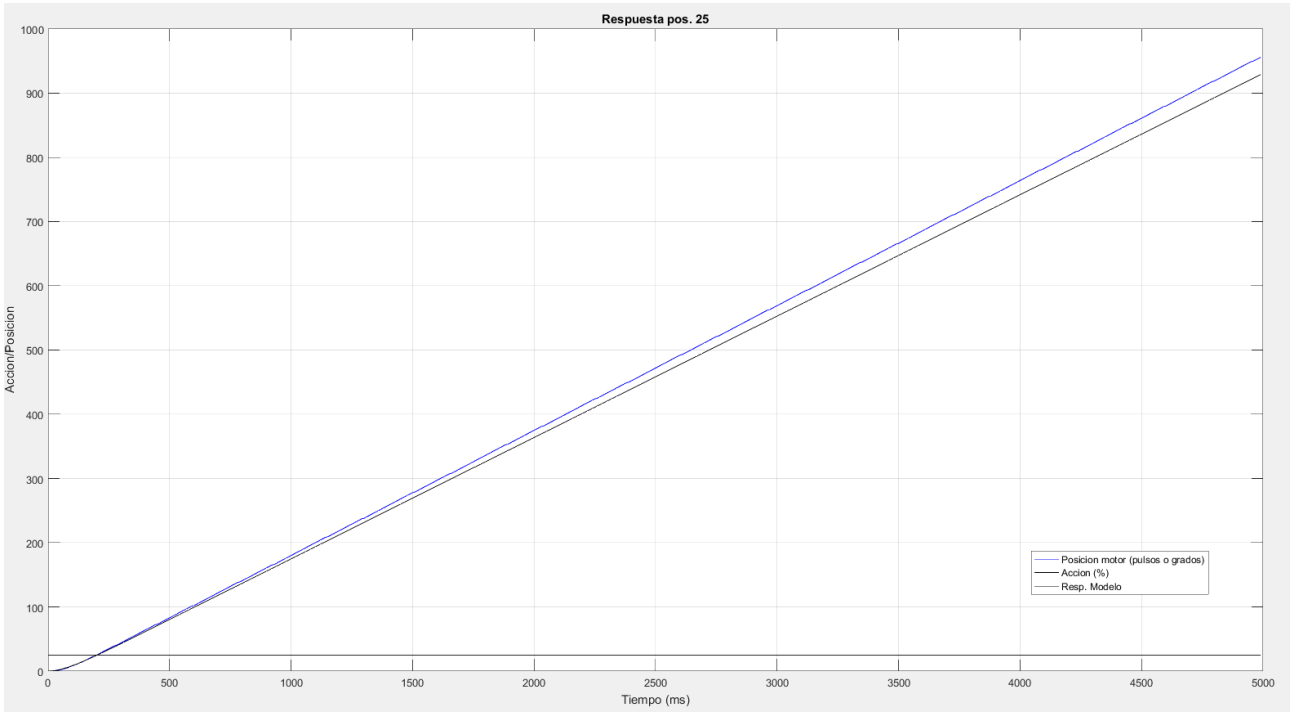


Figura 21. Resultado para entrada del 25%.

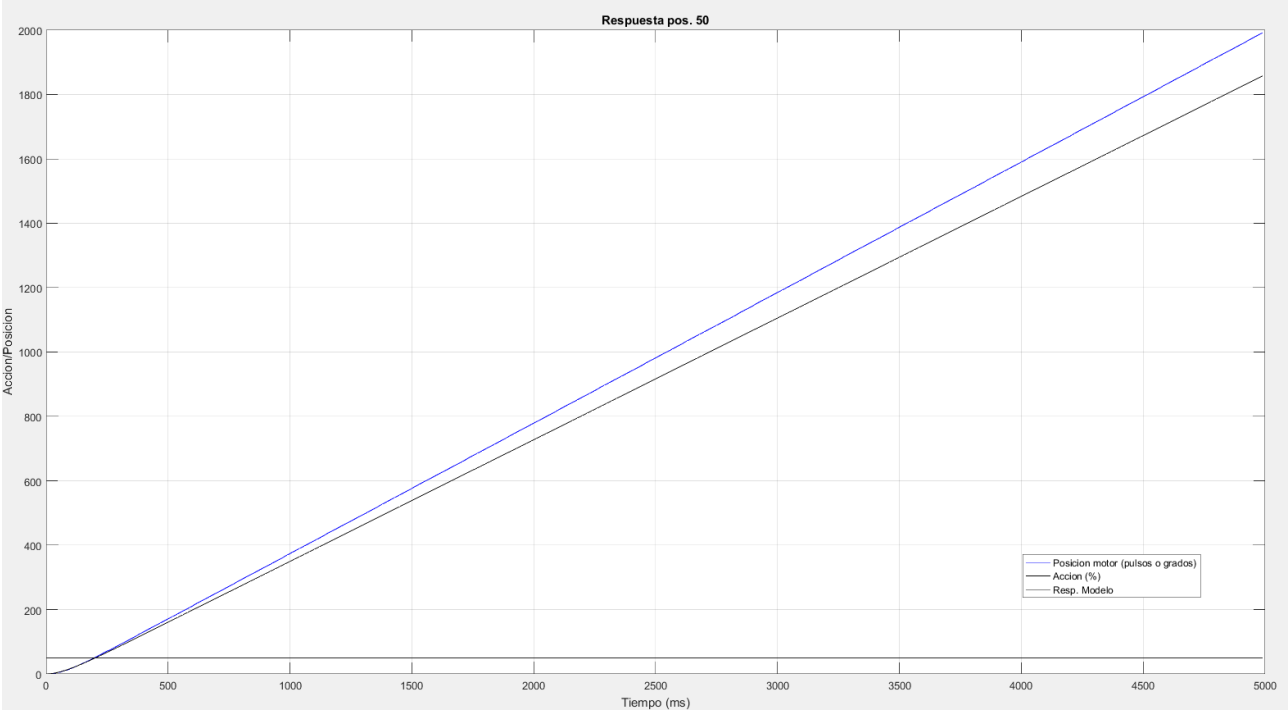


Figura 22. Resultado para entrada del 50%.

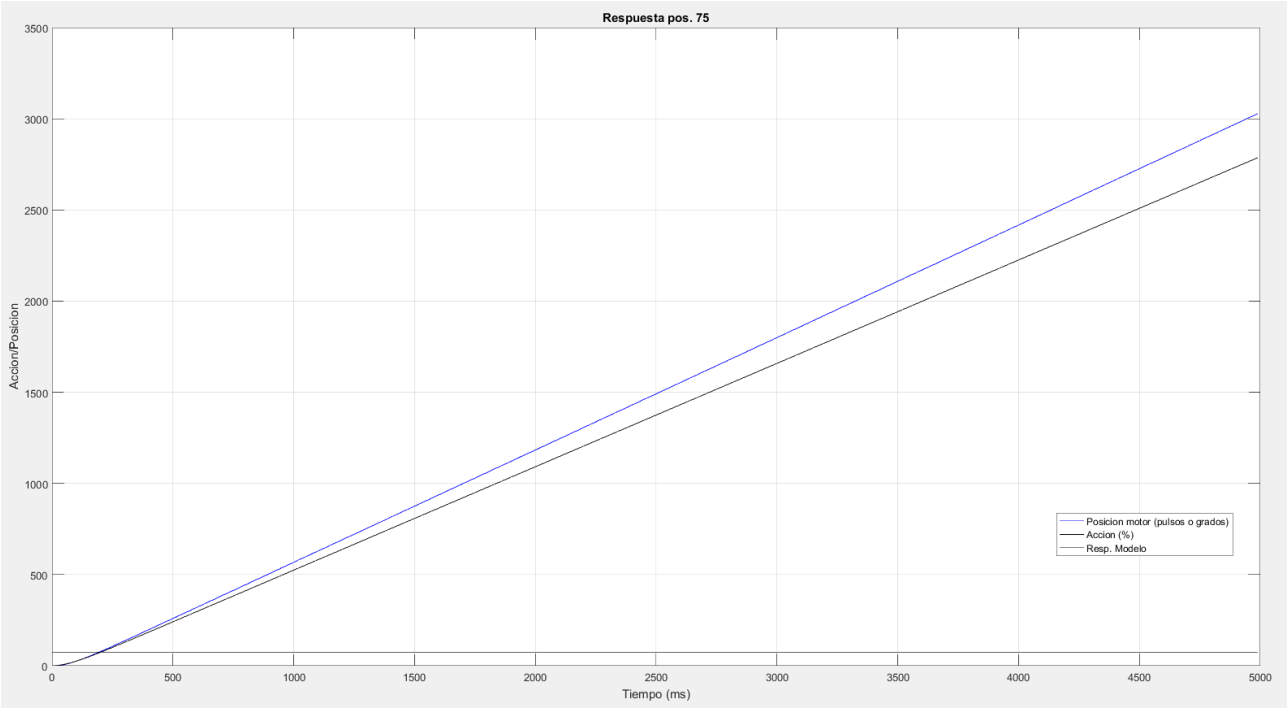


Figura 23. Resultado para entrada del 75%.

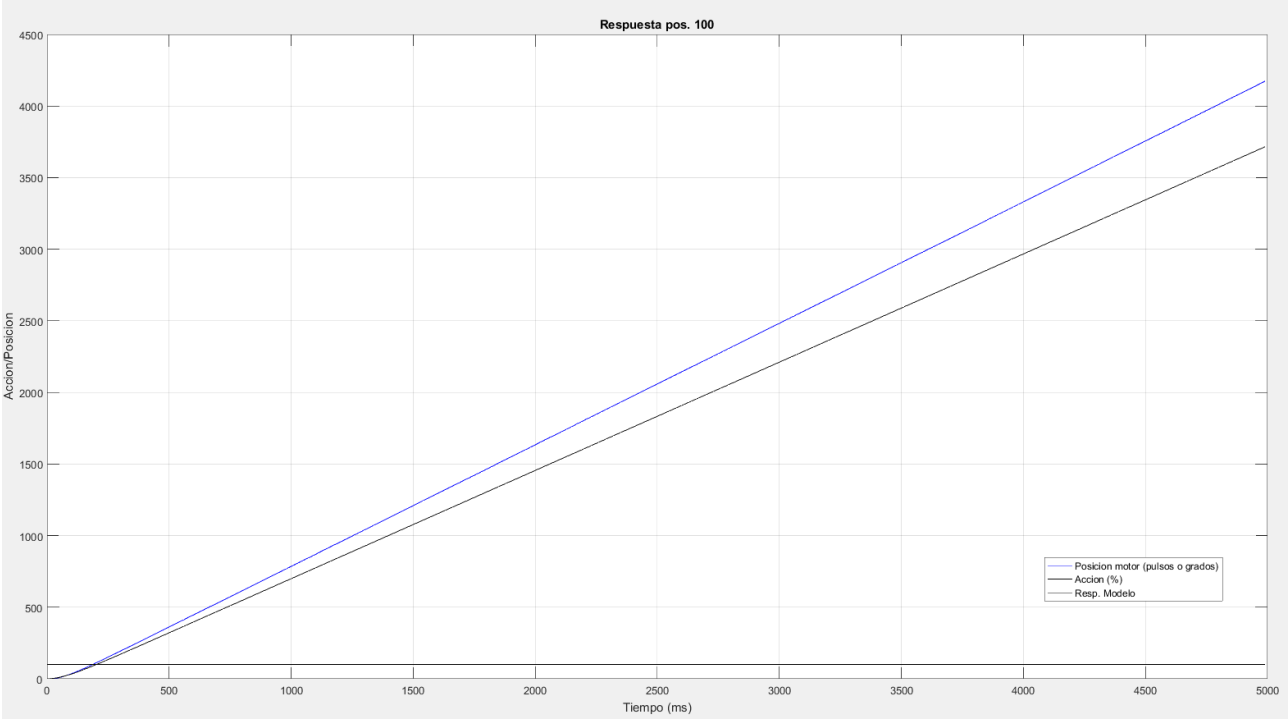


Figura 24. Resultado para entrada del 100%.

Donde se puede apreciar el seguimiento de la respuesta del modelo a la respuesta real junto con cierto error que podría atenuarse iterando una serie de ensayos de prueba y error, o utilizando un método más preciso que el gráfico para identificar los parámetros del modelo del motor como el análisis matemático de la serie de puntos (métodos de linealización).

Mediante una pequeña modificación del programa anterior podemos obtener un modelo para la velocidad en lugar de la posición. Para ello obtenemos el valor de posición actual, lo restamos al anterior y lo dividimos por el tiempo transcurrido (el periodo de muestreo). Un detalle a tener en cuenta es la zona muerta en la puesta en marcha del motor debido a las fuerzas inerciales que ha de vencer el eje para comenzar a girar como se ha explicado en la obtención matemática del modelo. Por ello, para evitar un falseamiento de los datos por la presencia de esta zona muerta, se dará un escalón de entrada y se esperará a que el sistema se estabilice antes de comenzar el muestreo con un nuevo escalón de distinto valor.

El código utilizado se incluye en el mismo anexo [ANEXO A1.1] debidamente comentado, como en el caso anterior.

De nuevo, con los datos obtenidos que se han guardado en un archivo de texto, y con el *script* de Matlab que se incluye en el anexo, realizamos la representación gráfica y la identificación que, a continuación, se desarrolla.

Para la velocidad con una entrada de escalón del ciclo de trabajo del 25% habiendo alcanzado una velocidad previa correspondiente a un ciclo de trabajo de un 15% para evitar la zona muerta:

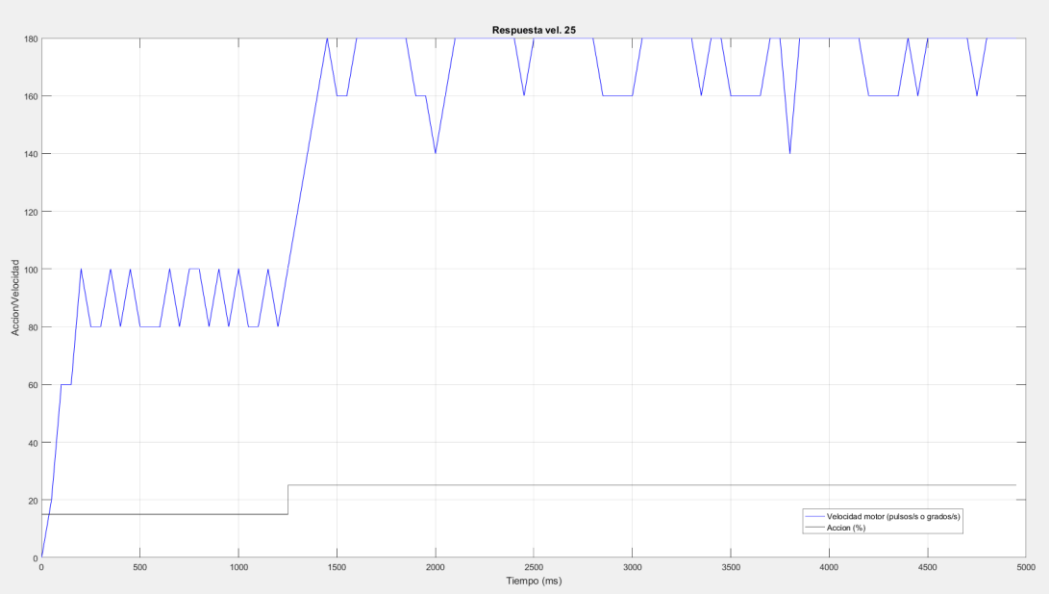


Figura 25. Respuesta de velocidad para entrada del 25%.

$$K_v = \frac{180}{25} = 7.2$$

$$0.632 \cdot 180 = 113.7$$

$$\tau_v = 50\ ms = 0.05s$$

Siendo:

$$G(s) = \frac{7{,}2}{1 + 0.05s}$$

Para la velocidad con una entrada de escalón del ciclo de trabajo del 50% habiendo alcanzado una velocidad previa correspondiente a un ciclo de trabajo de un 15% para evitar la zona muerta:

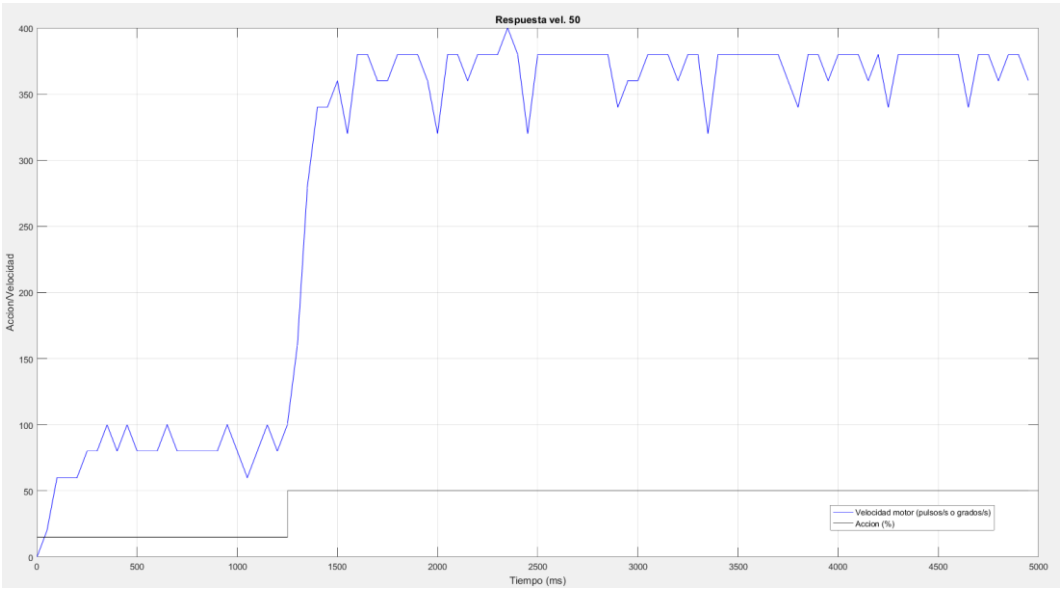


Figura 26. Respuesta de velocidad para entrada del 50%.

$$K_v = \frac{370}{50} = 7.4$$

$$0.632 \cdot 370 = 233.84$$

$$\tau_v = 80\ ms = 0.08\ s$$

Siendo:

$$G(s) = \frac{7{,}4}{1 + 0.08s}$$

Para la velocidad con una entrada de escalón del ciclo de trabajo del 75% habiendo alcanzado una velocidad previa correspondiente a un ciclo de trabajo de un 15% para evitar la zona muerta:

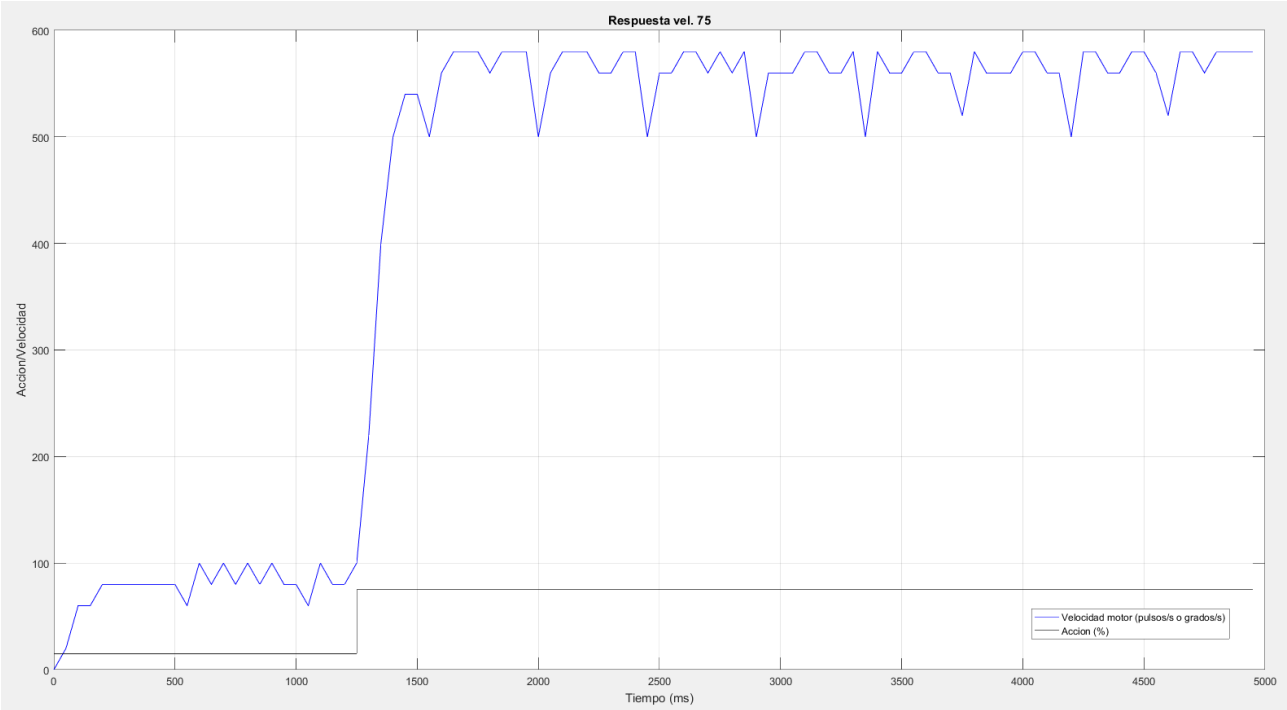


Figura 27. Respuesta de velocidad para entrada del 75%.

$$K_v = \frac{550}{75} = 7.3$$

$$0.632 \cdot 550 = 347.6$$

$$\tau_v = 85\ ms = 0.085\ s$$

Siendo:

$$G(s) = \frac{7{,}47}{1 + 0.085s}$$

Para la velocidad con una entrada de escalón del ciclo de trabajo del 100% habiendo alcanzado una velocidad previa correspondiente a un ciclo de trabajo de un 15% para evitar la zona muerta:

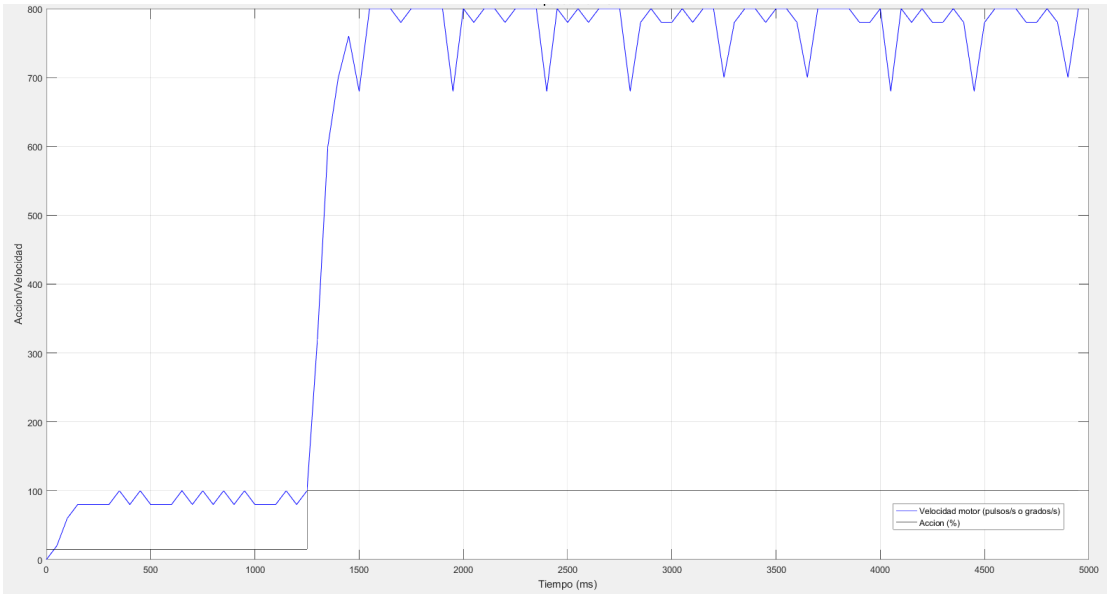


Figura 28. Respuesta de velocidad para entrada del 100%.

$$K_v = \frac{800}{100} = 8$$

$$0.632 \cdot 800 = 480.32$$

$$\tau_v = 83\ ms = 0.083\ s$$

Siendo:

$$G(s) = \frac{8}{1 + 0.083s}$$

Promediando como en el caso de la posición se obtiene:

$$G(s) = \frac{7.4}{1 + 0.075s}$$

Y mediante el esquema de Simulink que a continuación presentamos, podemos validar el modelo:

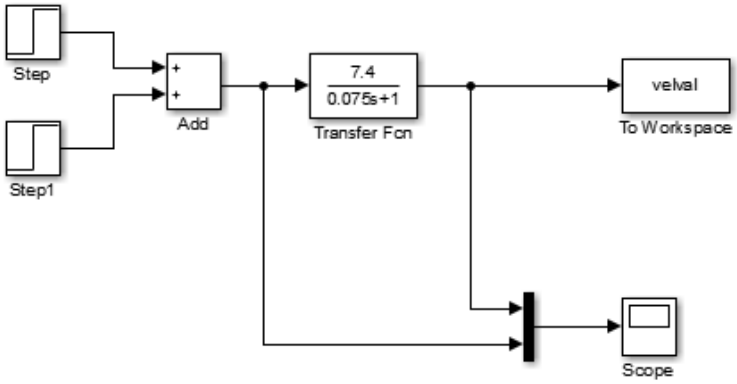


Figura 29. Esquema para la comprobación del modelo de velocidad.

Para 25%:

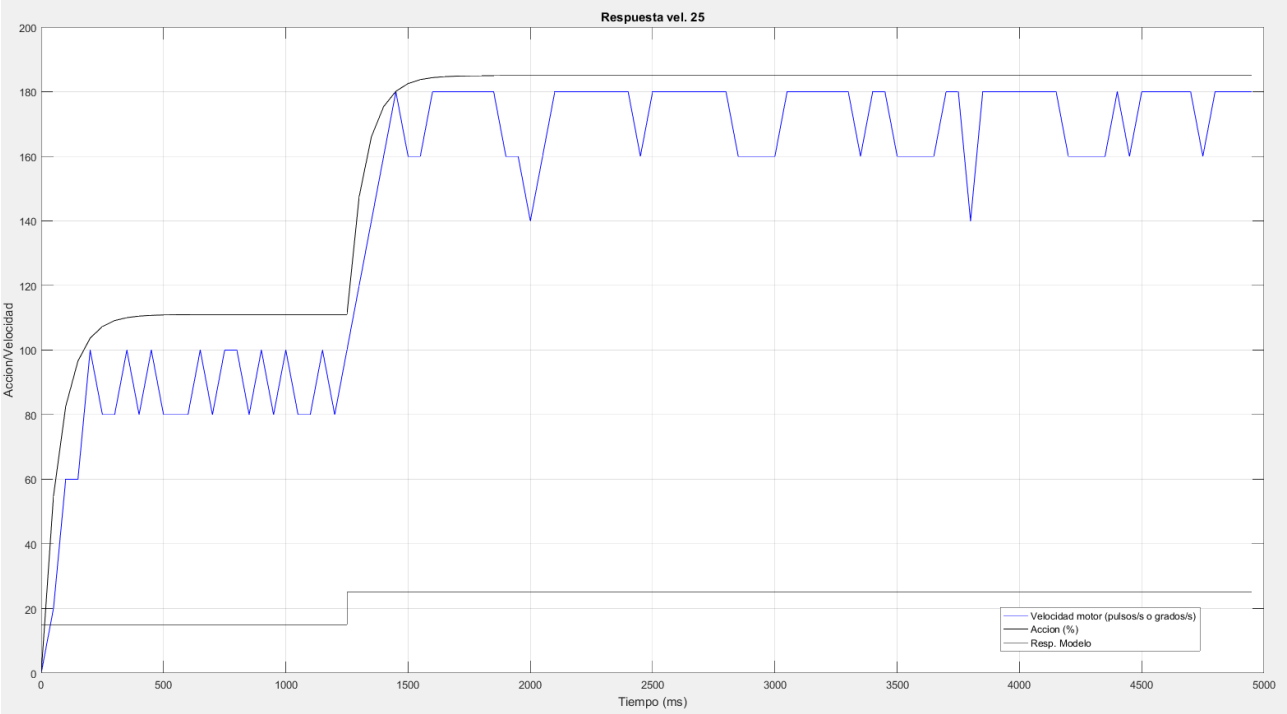


Figura 30. Respuesta del modelo de velocidad para entrada del 25%.

Para 50%:

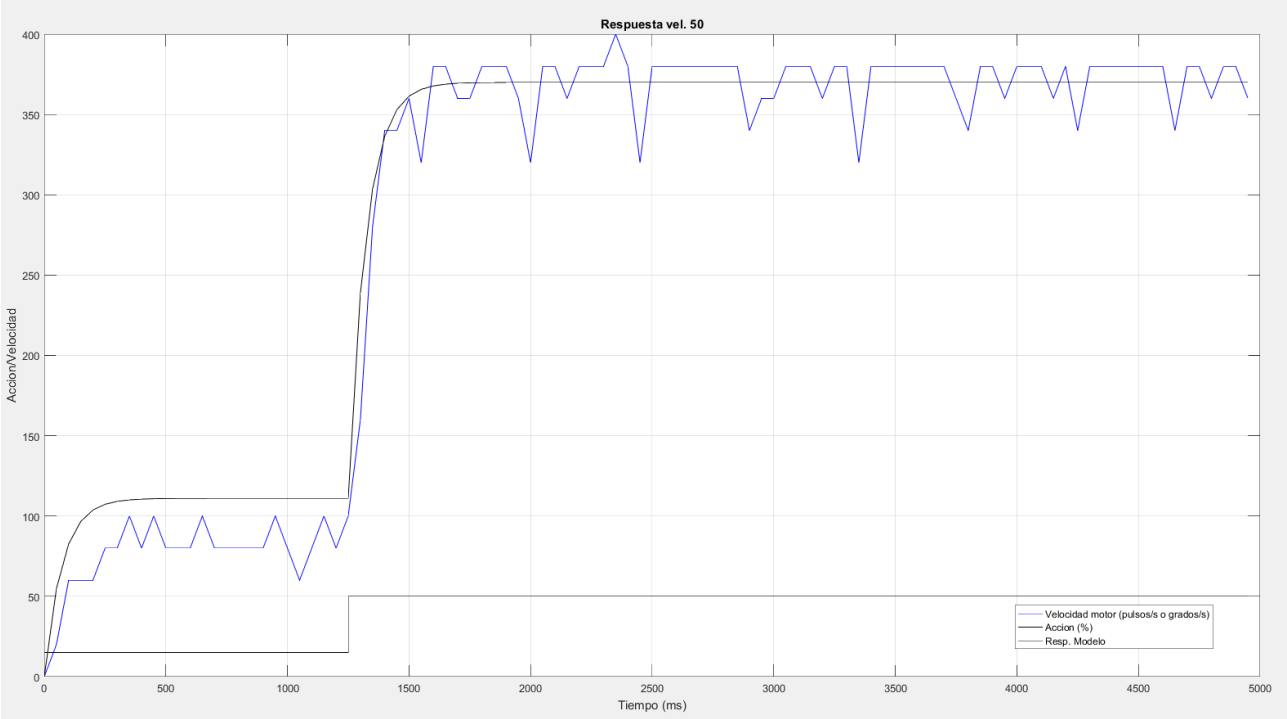


Figura 31. Respuesta del modelo de velocidad para entrada del 50%.

Para 75%:

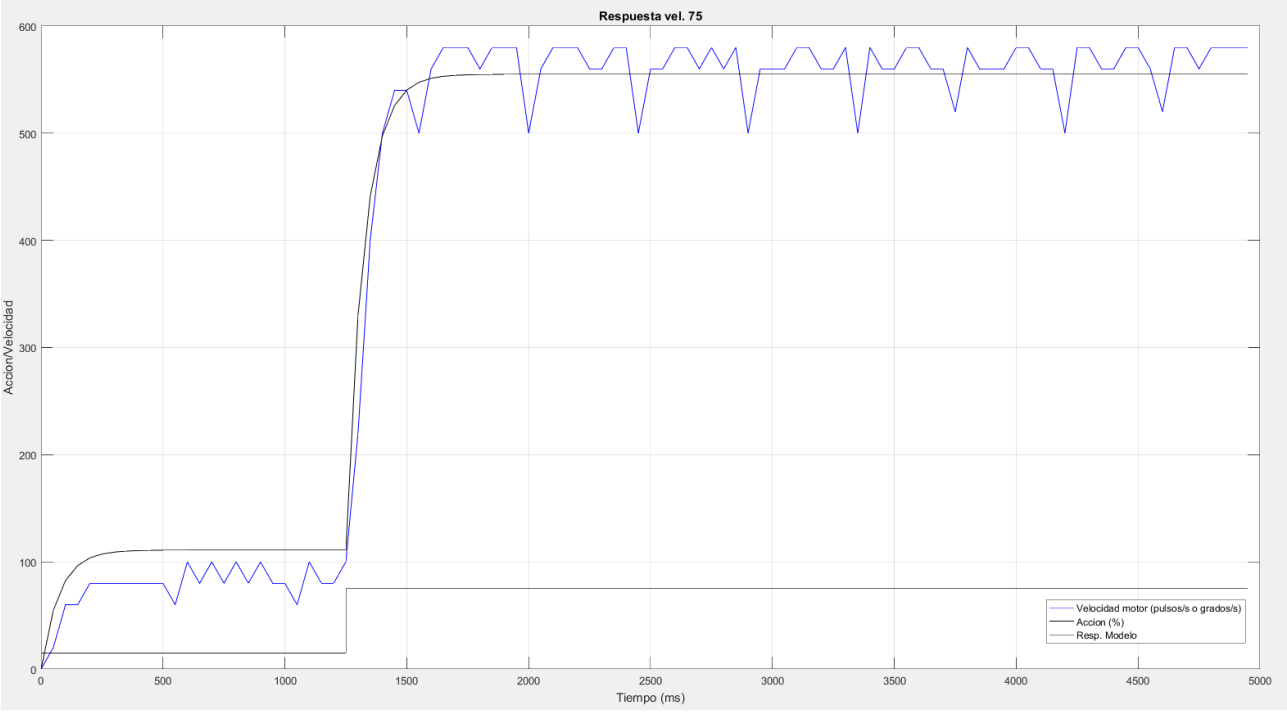


Figura 32. Respuesta del modelo de velocidad para entrada del 75%.

Para 100%:

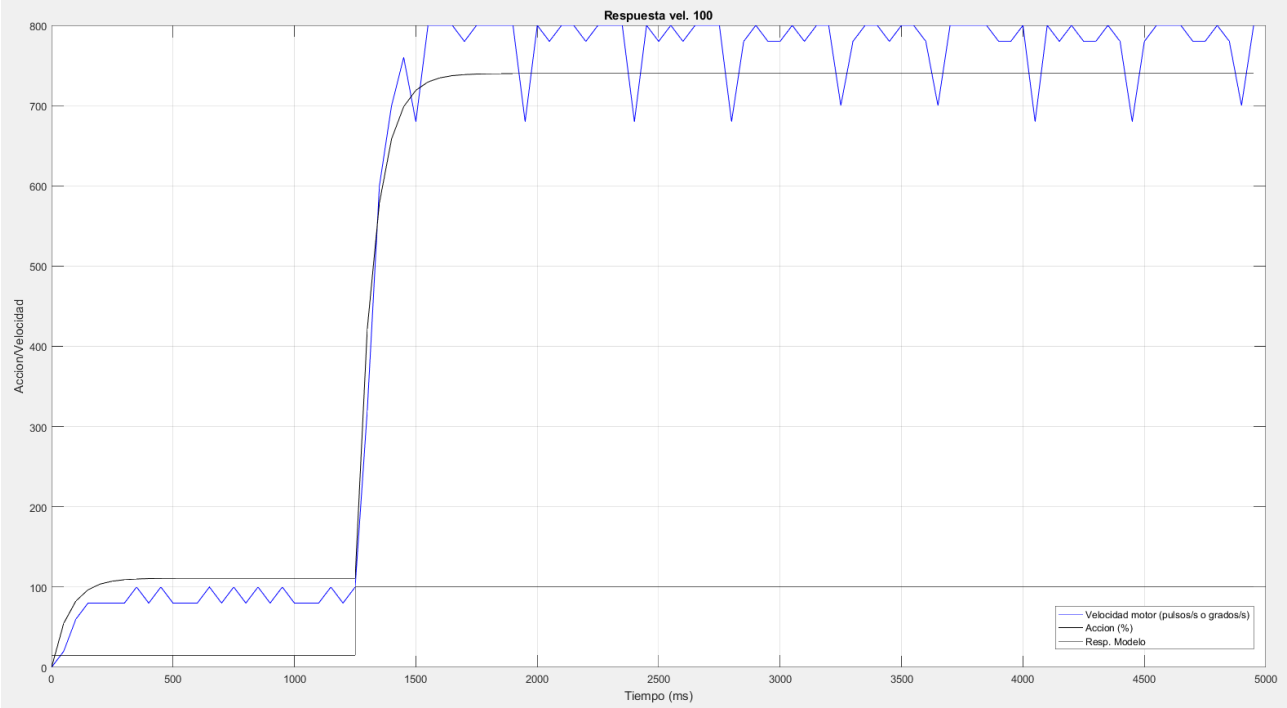


Figura 33. Respuesta del modelo de velocidad para entrada del 100%.

La salida sigue a la respuesta real del sistema cumpliendo con los criterios de tiempo y ganancia.

Para el método de obtención del modelo a partir de la obtención de la ganancia y la constante de tiempo debemos discretizar la función de transferencia obtenida.

En cualquier caso, obtenida la función de transferencia discreta, para simular la respuesta del sistema trabajaremos con la ecuación en diferencias.

Ahora discretizaremos ambos modelos y observaremos la respuesta respecto al sistema real para validarlos. Un forma de conseguir la función de transferencia discreta es mediante el comando *c2dm* (paso de continuo a discreto con un método concreto) en Matlab. Para ello se definirían dos polinomios, numerador y denominador y se llamaría a la función como:

[num_dis, den_dis] = c2dm(num_cont, den_cont, Ts, 'zoh')

Donde el tercer parámetro que se le pasa a la función es el periodo de muestreo y el cuarto es el método de discretización. En este caso, por tratarse de una función de transferencia de un proceso, se utilizará un retenedor de orden cero (Zero Order Hold).

Otra forma de discretizar es mediante la tabla de relación entre el plano S y Z, que para un sistema de primer orden con retenedor de orden cero tendría una equivalencia:

$$G_P(s) = \frac{b}{s+a} \rightarrow G_P(z) = \frac{b}{a}\frac{(1-e^{aT})}{z-e^{aT}}$$

En el mismo anexo que en los casos anteriores se incluye el *script* de Matlab que realiza esta discretización. Las funciones de transferencia discretas que se obtienen son:

Para la posición:

$$G(Z) = \frac{0.0048z + 0.0046}{z^2 - 1.8752z + 0.8752}$$

Para la velocidad:

$$G(Z) = \frac{0.9237}{z - 0.8752}$$

Se puede ahora representar la respuesta del modelo discreto, comparándola con la del modelo continuo y la obtenida experimentalmente. Ilustraremos esta comparación solo para un caso de los cuatro estudiados siendo el resultado equivalente para el resto.

Para la posición, con el esquema:

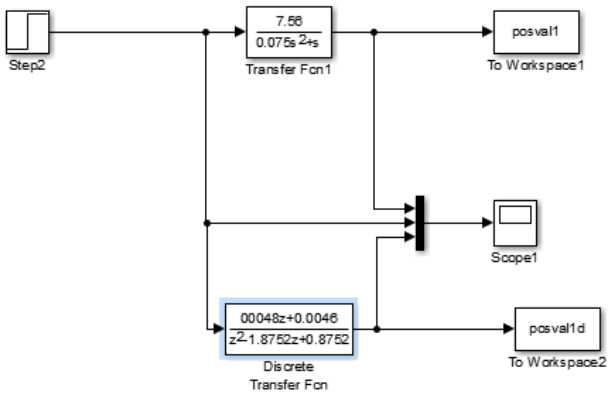


Figura 34. Esquema para la simulación del modelo discreto de posición

Para el ciclo de trabajo del 100%:

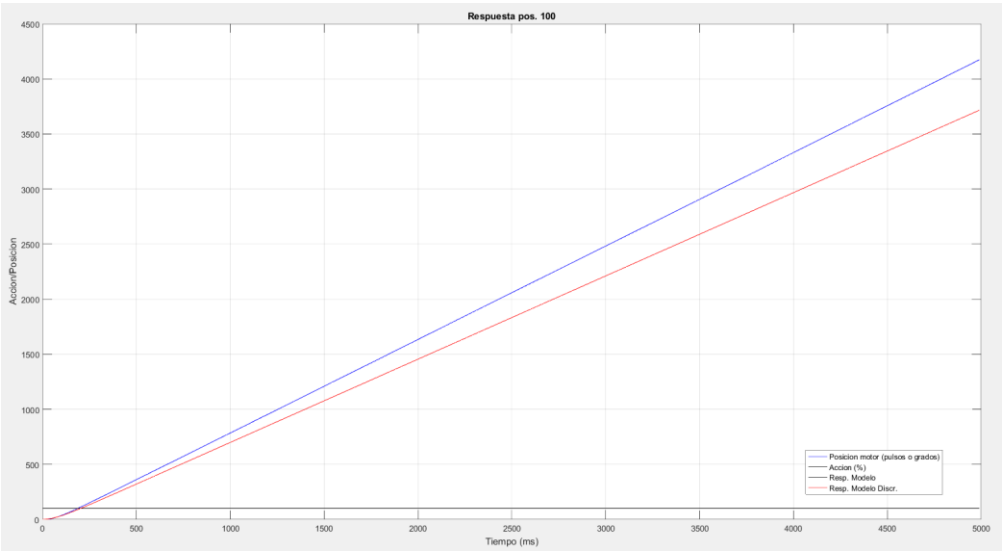


Figura 35. Respuesta del modelo discreto de posición para entrada del 100%.

Para la velocidad, con el esquema:

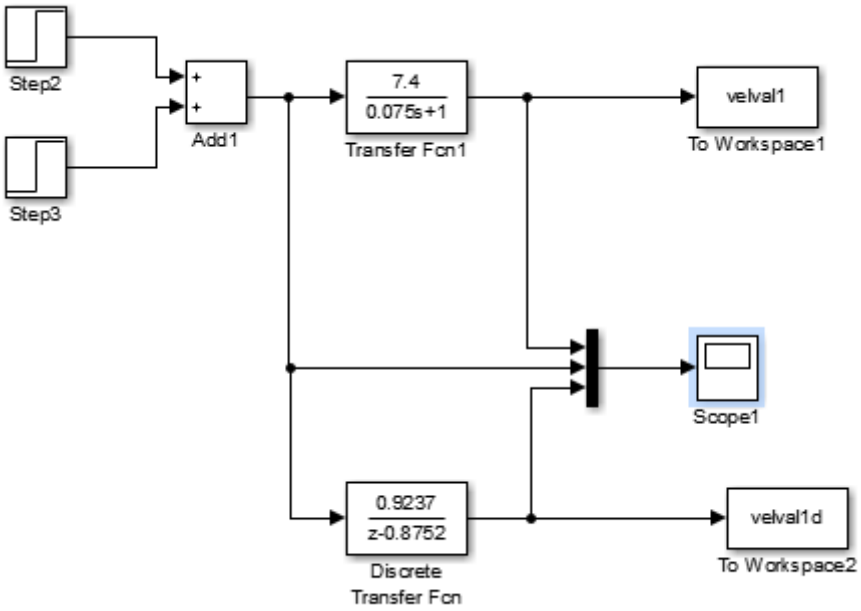


Figura 36. Esquema para la simulación del modelo discreto de velocidad

Para el ciclo de trabajo del 100%:

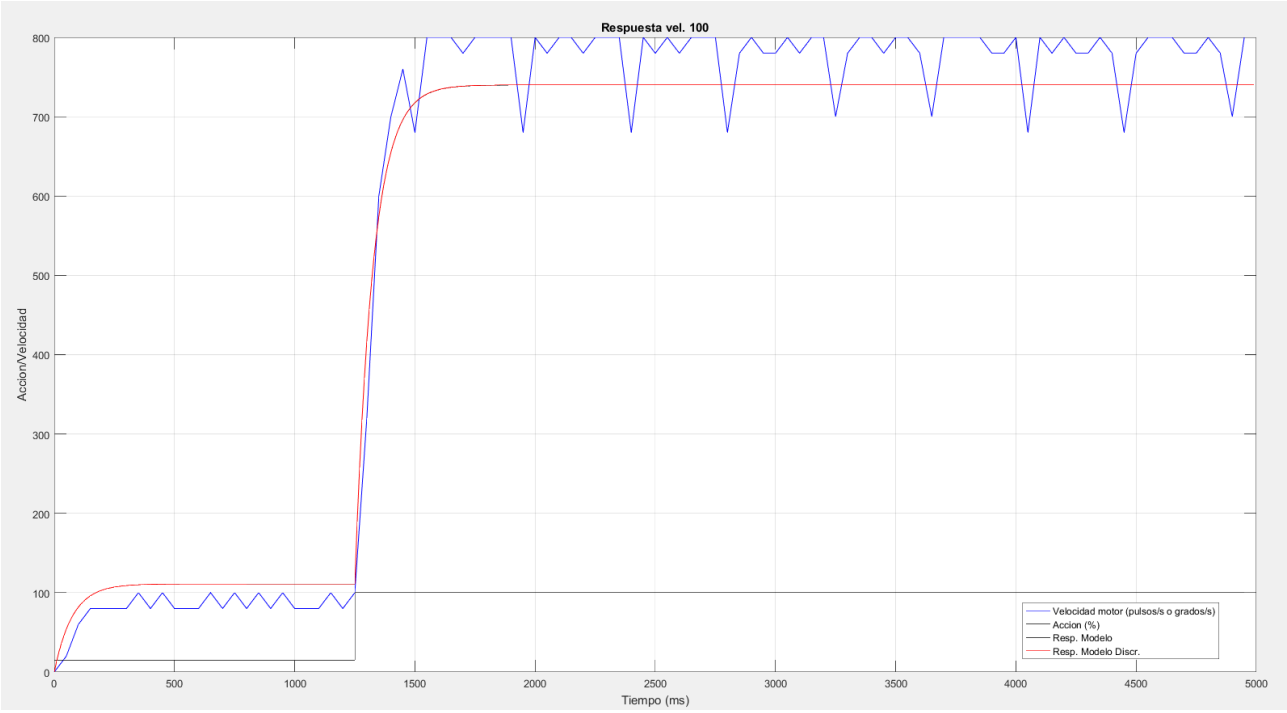


Figura 37. Respuesta del modelo discreto de velocidad para entrada del 100%.

Como se puede apreciar, este modelo "respeta" la respuesta del sistema en todo momento, si bien para la posición la rampa presenta una pendiente algo diferente. Este problema se podría remediar reduciendo la ganancia de la función de transferencia del modelo, que es el parámetro relacionado con la pendiente de la respuesta experimental del motor. Se puede, no obstante, tomar

este modelo como válido, tanto para posición como para velocidad, pues se ajusta suficientemente al comportamiento real del motor y no supone una limitación de mayor trascendencia en el desarrollo del presente trabajo.

4.3.3 Modelado paramétrico. Estimación mediante aproximación por mínimos cuadrados

El método de aproximación por mínimos cuadrados que seguiremos es otra de las maneras de llegar al modelo siguiendo el camino experimental. Consiste en una aproximación matemática buscando reducir el error entre la respuesta real del sistema y la del modelo. El planteamiento es:

Dado un sistema genérico con una función de transferencia discreta:

$$G(z) = \frac{\hat{Y}(z)}{U(z)} = \frac{b_0 + b_1 z^{-1} + \cdots + b_m z^{-m}}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_n z^{-n}} z^{-d}$$

Ecuación 29. Función de transferencia discreta genérica.

La expresión en ecuaciones en diferencias será:

$$\hat{y}_k = b_0 u_{k-d} + b_1 u_{k-d-1} + \cdots + b_m u_{k-d-m} + a_1 \hat{y}_{k-1} + \cdots + a_n \hat{y}_{k-n}$$

Ecuación 30. Ecuación en diferencias genérica.

Denominaremos error a la diferencias entre la salida real del sistema y la del modelo:

$$e_k = y_k - \hat{y}_k$$

Expresado en forma matricial:

$$\begin{bmatrix} e_{k+1} \\ e_{k+2} \\ \cdots \\ e_{k+N} \end{bmatrix} = \begin{bmatrix} y_{k+1} \\ y_{k+2} \\ \cdots \\ y_{k+N} \end{bmatrix} - \begin{bmatrix} -\hat{y}_k & -\cdots & -\hat{y}_{k+1-n} & u_{k+1-d} & \cdots & u_{k+1-d-m} \\ -\hat{y}_{k-1} & -\cdots & -\hat{y}_{k+1-n} & u_{k+2-d} & \cdots & u_{k+2-d-m} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ -\hat{y}_{k+N-1} & -\cdots & -\hat{y}_{k+1-n} & u_{k+N-d} & \cdots & u_{k+N-d-m} \end{bmatrix} \begin{bmatrix} a_1 \\ \cdots \\ a_n \\ b_m \\ \cdots \\ b_m \end{bmatrix}$$

$$E(N, \Theta) = Y(N) - \Phi(N)\widehat{\Theta}$$

Buscamos minimizar:

$$J(\Theta) = \frac{1}{2}\sum_{i=1}^{N} e_{k+i}^2 = \frac{1}{2} E(N, \Theta)^T E(N, \Theta)$$

Obteniendo nuestro modelo a partir de los coeficientes de la matriz al despejar:

$$\widehat{\Theta} = (\Phi(N)^T \Phi(N))^{-1} \Phi(N)^T \mathrm{Y}(N)$$

Ecuación 31. Ecuación de la obtención de la matriz de coeficientes

Se plantean una serie de inconvenientes a la hora de realizar la aproximación. El primero es la

necesidad de una gran cantidad de medidas para minimizar el efecto de los ruidos y las perturbaciones durante los ensayos. El segundo, el problema de cálculo, necesitamos invertir una matriz demasiado grande. Y el tercero, no es un método adecuado para sistemas con dinámica cambiante a lo largo del ensayo.

Por todo esto, se llega a una variación del método, el de mínimos cuadrados recursivo. Esta variante utiliza las medidas conforme las va recogiendo realizando las operaciones necesarias en cada instante lo que reduce drásticamente la carga de cálculo. Además, es apropiado para sistemas cuya dinámica cambie durante la prueba. La nueva idea es:

$$\widehat{\Theta}(k) = \widehat{\Theta}(k-1) + K(k)(y(k) - \Phi(k)\widehat{\Theta}(k-1))$$

Donde la ganancia de adaptación es:

$$K(k) = P(k)\Phi(k)^T$$

Y $P(k) = (\Phi(k)^T\Phi(k))^{-1}$ es definida positiva.

De manera que dando valores iniciales a P y θ (P debe tener inicialmente un valor elevado), en cada instante k de nuestro programa, haríamos:

Leer valores de y(k) y u(k).

Formar el vector regresor $\Phi(k)$.

Calcular $P(k)$ como: $P(k) = P(k-1) - \frac{P(k-1)\Phi(k)^T\Phi(k)P(k-1)}{1-\Phi(k)P(k-1)\Phi(k)^T}$

Calcular $K(k)$ como: $K(k) = \frac{P(k-1)\Phi(k)^T}{1-\Phi(k)P(k-1)\Phi(k)^T}$

Calcular $\widehat{\Theta}(k)$ como: $\widehat{\Theta}(k) = \widehat{\Theta}(k-1) + K(k)(y(k) - \Phi(k)\widehat{\Theta}(k-1))$

La implementación práctica de estos métodos requiere de una entrada de excitación persistente basada en una señal PRBS (Pseudo-Random Binary Sequence), una señal que oscila entre dos valores con un ciclo de trabajo aleatorio. Nuestra señal PWM puede hacer las veces de PRBS si tenemos en cuenta que el valor medio proporcionado por el ciclo de trabajo oscila en torno al valor elegido. Otra característica a tener en cuenta es la zona muerta que posee el motor debido a la resistencia e inercias mecánicas que tiene que vencer el eje para comenzar el movimiento, por ello primero daremos un valor de velocidad al motor y cuando se estabilice comenzaremos la prueba. Ya que disponemos de una potente herramienta matemática como es Matlab, utilizaremos el primer método para obtener el modelo.

En el anexo [ANEXO A1.6] es encuentra el código de la función para generar la señal PRBS y la traducirla a nuestra acción de control.

En el anexo [ANEXO A1.2] tenemos el código utilizado para el ensayo, así como el script de Matlab que implementa el algoritmo de mínimos cuadrados para la obtención de los parámetros del sistema. Como resultado tenemos:

Para la posición, partiendo del modelo discreto genérico como el obtenido en el punto anterior:

$$G(z) = \frac{(b_0 + b_1 z^{-1})}{(1 + a_1 z^{-1} + a_2 z^{-2})} z^{-d} = \frac{(b_0 z + b_1)}{(z^2 + a_1 z + a_2)} z^{-(d-1)}$$

$$d = 4$$

$$a_1 = -0.8584$$

$$a_2 = -0.1416$$

$$b_0 = 0.0474$$

$$b_1 = 0.0487$$

$$G(z) = \frac{(0.0474 + 0.0487 z^{-1})}{(1 - 0.8584 z^{-1} - 0.1416 z^{-2})} z^{-4} = \frac{(0.0474 z + 0.0487)}{(z^2 - 0.8584 z - 0.1416)} z^{-3}$$

Podemos validar el modelo del sistema con el siguiente esquema de Simulink:

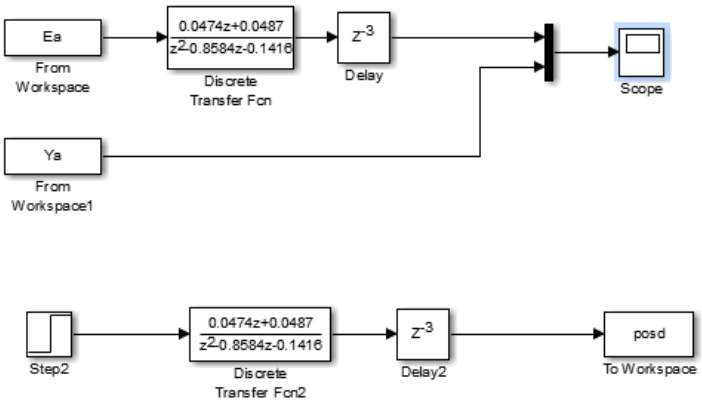


Figura 38. Esquema para la validación del modelo discreto paramétrico

Obteniendo como resultados al compararlo con la respuesta ante escalón de entrada de 50% del ciclo de trabajo del modelado anterior:

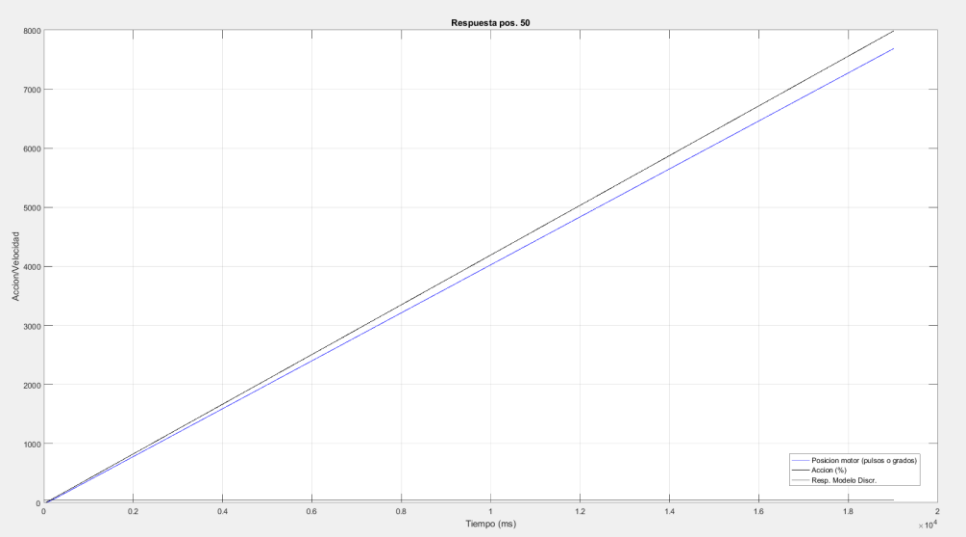


Figura 39. Respuesta del modelo de posición ante entrada de 50%

Donde sí se puede apreciar una notable mejora respecto al método de modelado del apartado anterior en el seguimiento de la señal experimental.

Para la velocidad, partiendo del modelo discreto de primer orden:

$$G(z) = \frac{b}{1 + az^{-1}} z^{-d}$$

$$a = -0.0838$$

$$b = 6.0272$$

$$d = 4$$

$$G(z) = \frac{6.0272}{1 - 0.0838z^{-1}} z^{-4} = \frac{6.0272}{z - 0.0838} z^{-3}$$

Podemos validar el modelo del sistema con el siguiente esquema de Simulink:

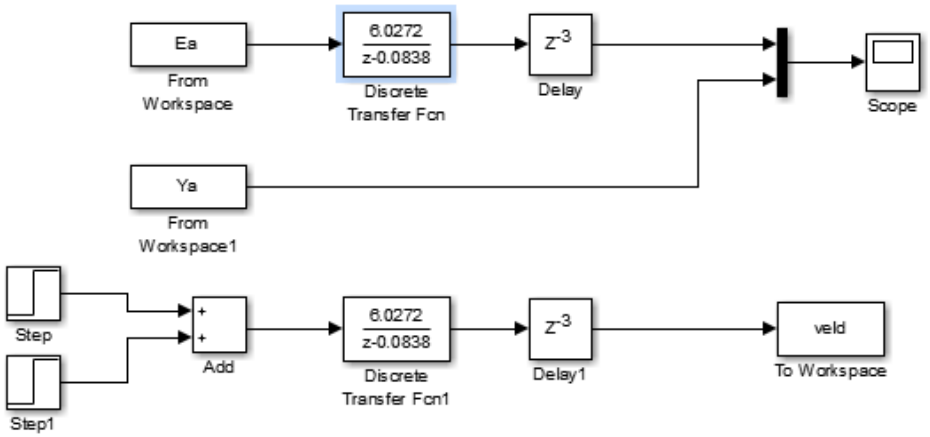


Figura 40. Esquema para la validación del modelo discreto paramétrico

Obteniendo como resultados al compararlo con la respuesta ante escalón de entrada de 50% del ciclo de trabajo del modelado anterior:

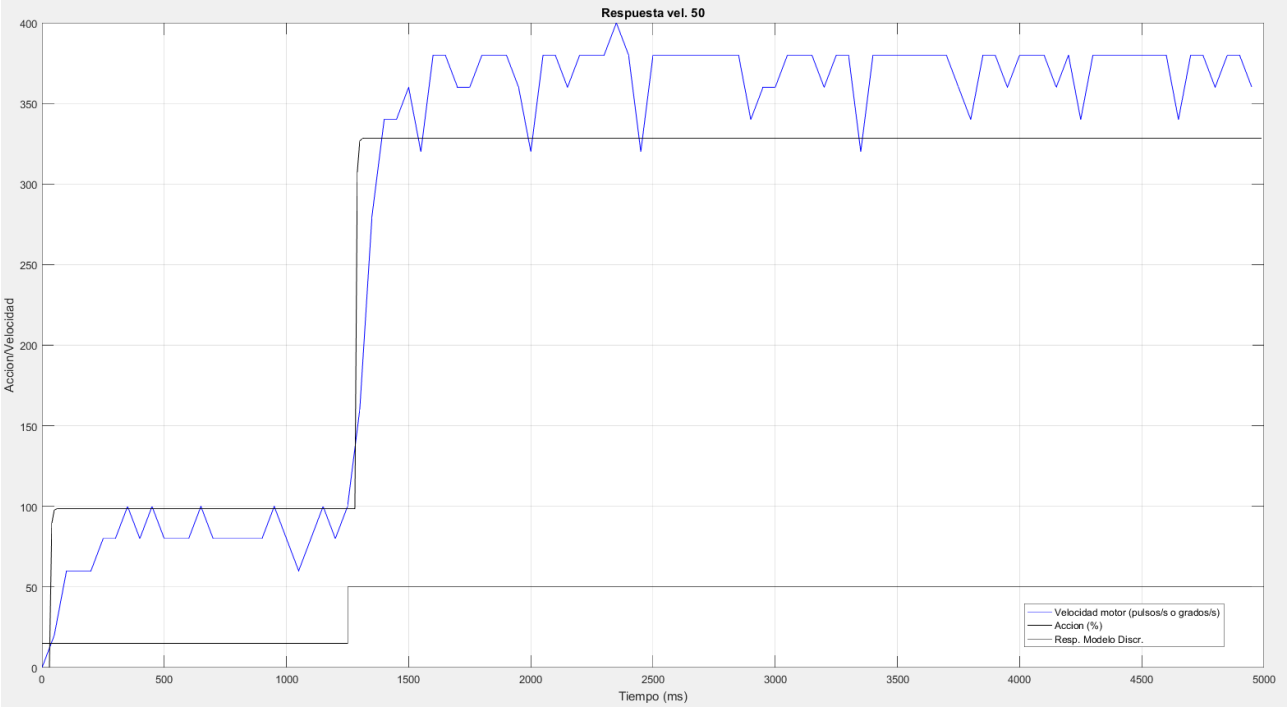


Figura 41. Respuesta del modelo de velocidad ante entrada de 50%

Donde se comprueba el seguimiento del modelo a la respuesta real del sistema. No obstante, este modelo presenta peor seguimiento, como hemos podido demostrar, para esta configuración de sistema.

Pese a la mejora que se produce en el modelo de la posición, se utilizará los dos modelos obtenidos en el apartado anterior mediante comparación con la respuesta ante escalón para el diseño del control.

4.4 Elección del sensor

La elección del sensor determinará las características de la señal de salida que realimentaremos en nuestro control. Estudiaremos diferentes alternativas para cada una de las variables del sistema que necesitamos medir.

4.4.1 Medida de la posición

Esta medida será la que nos indique la posición del eje de nuestro motor en cada instante de tiempo. Existen al menos tres posibilidades que se suelen implementar como soluciones a nivel industrial:

- Potenciometros: estos suponen una solución sencilla y económica que no requiere de componentes adicionales. La posición del eje se traduce en un valor de tensión analógico entre la referencia y la alimentación, dado por una resistencia variable unida a dicho eje. Se puede obtener una gran precisión trabajando con potenciómetros multivuelta de última tecnología. Sin embargo, esta solución presenta como inconveniente principal la limitación de velocidad de funcionamiento a la que se podría obtener un valor fiable de medida. Además, presenta problemas de ruido eléctrico, y de desgaste, y requieren un conversor A/D para su lectura.

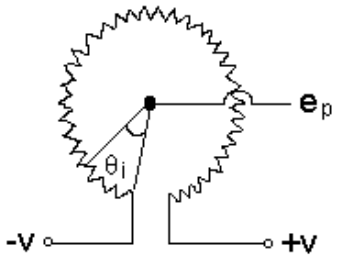


Figura 42. Esquema de funcionamiento de potenciometro.

- Encoders: sin lugar a dudas la solución más utilizada a nivel industrial y comercial. Los encoders o codificadores digitales consisten en unos discos marcados codificados y un sensor, normalmente formado por un emisor led y un fotodetector, que indica mediante pulsos la detección de las marcas o ranuras del disco al girar de forma solidaria con el eje del motor.

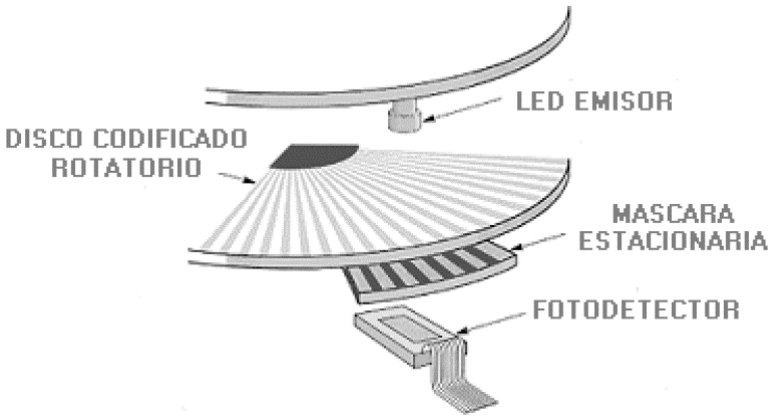


Figura 43. Modelo de un encoder [SR 15].

Existen diferentes configuraciones de encoder, siendo los incrementales los más utilizados, constituidos por dos discos cuyas marcas están desfasadas 90º. La lectura por separado de cada uno de los discos permite comparar el orden de aparición de las marcas y, con ello, la obtención del sentido de giro.

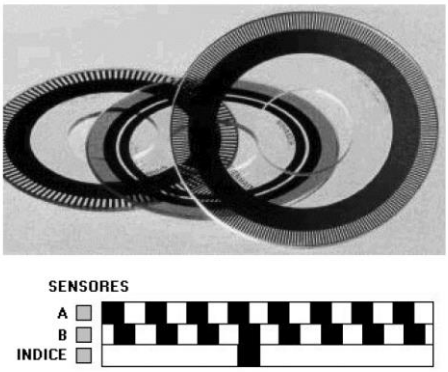


Figura 44. Codificación de encoder incremental

La otra configuración de encoder son los absolutos. Estos se encuentran completamente codificados en varios bits de resolución, de forma que se conozca la posición en todo momento entre 0 y 360 grados del eje.

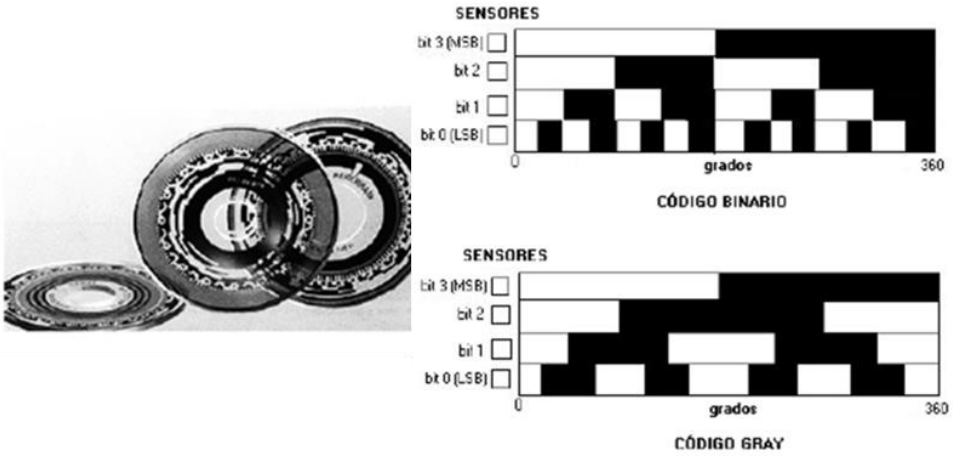


Figura 45. Codificación de encoder absoluto de 4 bits.

Si bien los encoders implican ligereza y alta resolución, son poco robustos mecánicamente, lo que los hace sensibles a golpes y vibraciones.

- Resolvers: estos elementos se sirven de un embobinado de excitación, que gira junto al eje del motor, y dos embobinados bifásicos separados 90º que son excitadas por el primero. De esta forma al inducir una corriente senoidal en el embobinado de excitación en el rotor, esta se induce en las bobinas que hacen de sensor produciendo un seno y un coseno como corriente de respuesta.

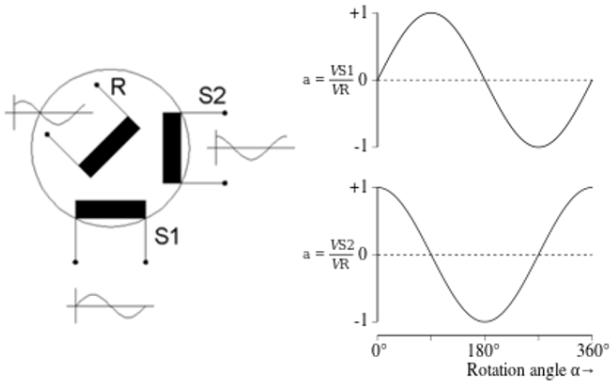


Figura 46. Modelo y señales del resolver.

Los resolvers poseen una gran robustez mecánica, sin embargo resultan menos precisos que los encoders, son sensibles al ruido, requieren de circuitería adicional para interpretar las señales y presentan un retardo.

Para este proyecto, en el caso de la medida de la posición, se ha elegido un encoder incremental por las ventajas que presenta respecto al resto y su bajo coste. Nuestro sensor trabajará mandándonos pulsos para indicar la posición que nosotros contaremos continuamente mediante una interrupción gestionada por software asociada a un timer, por lo que no será necesario un convertidor A/D. Veremos con más detenimiento la implementación práctica más adelante.

4.4.2 Medida de la velocidad

La medida de la velocidad del eje del motor será imprescindible para el seguimiento de nuestro perfil de movimiento. Existen varias posibilidades para la obtención del valor de velocidad, una de

las más empleadas es el uso de una tacodínamo.

Las tacodínamos proporcionan una señal de corriente continua con el giro del eje del motor. Están constituidas por un inductor que genera un campo magnético mediante imanes permanentes o electroimanes y un inducido o rotor ranurado sobre el que se bobinan unos devanados de hilo conductor. Suelen tener una sensibilidad entre 5 y 10 mV por cada r.p.m. y pueden medir velocidades de hasta 10.000 r.p.m.

Sin embargo, ya que se ha decidido utilizar un *encoder* incremental para la medida de posición del eje del motor, podemos aprovechar el mismo dispositivo para medir la velocidad simplemente dividiendo la diferencia entre dos valores de posición consecutivos por el tiempo entre medidas. De esta manera tenemos una solución sencilla y de bajo coste para la medida de dos de las tres variables necesarias para el control.

4.4.3 Medida de la intensidad

La medida de la intensidad que circula por el motor en cada instante, y que es un índice del par que éste realiza, es una tarea complicada. Existen multitud de posibilidades desde sensores de efecto Hall hasta bobinas de Rogowski pero la mayoría de ellas resultan poco factibles y complejas para las magnitudes de trabajo y la aplicación del proyecto. Por ello se ha optado por una solución más sencilla, la cual es posible gracias a nuestra etapa de potencia.

Para la medida de la intensidad del motor, utilizaremos una salida del L298 (nuestra etapa de potencia) con una resistencia para transformar la medida de intensidad en tensión que podamos medir con nuestra placa en un pin y un conversor A/D gestionado por DMA de manera que no utilice procesador para realizar la medición y conversión. Una vez más, lo veremos con mayor detalle en la implementación más adelante.

4.5 Elección del sistema de adquisición de datos

Tal y como se indicaba en el punto 3.6, el sistema de adquisición de datos es el encargado de tomar y administrar los datos, realizar los cálculos pertinentes y actualizar las salidas. Por ello incluye el procesador, la memoria y los periféricos necesarios para realizar conversiones (A/D y D/A) y administrar el tiempo entre procesos. Es, en definitiva, el cerebro que gobierna la aplicación.

En la actualidad existen cada vez una mayor cantidad de dispositivos que implementan un procesador (fuera de un computador completo) y todos estos componentes en un mismo elemento: el microcontrolador. Hoy en día casi cualquier elemento electrónico consiste en un sistema embebido cuyo cerebro es un microcontrolador. Por ello el abanico de posibilidades es muy amplio. Vamos a estudiar sólo unos pocos como posibles alternativas.

4.5.1 Tarjetas de adquisición de datos y PC

Una de las posibilidades más utilizadas para la disposición de puestos de control junto al proceso es el uso de tarjetas de adquisición de datos como interfaz entre el proceso y un computador. Existen en concreto dos tarjetas bastante utilizadas para la implementación de estos sistemas y con compatibilidad con Windows:

- PCI-9112 de ADLINK.

Esta primera tarjeta se instala directamente en la placa base del ordenador como si de una tarjeta gráfica o de sonido se tratara y permite la conexión al equipo mediante entrada de PCI-Bus de 32 Bits de hasta 16 canales de entrada analógicos con resolución de 12 bits, 2 canales de salida D/A con la misma resolución, contadores de 16 bits,etc.

Además dispone de una biblioteca de funciones en C para la programación de todas las funciones que se pudiera necesitar.

- NI USB 6008 de National Instruments.

Sin necesidad de instalación directa en el equipo, esta tarjeta de adquisición puede conectarse al ordenador mediante un puerto USB. Dispone de 8 entradas analógicas de 12 a 14 bits, 2 salidas analógicas temporizadas por software y un contador de 23 bits a 5MHz.

Figura 47. Modelo y señales del resolver.

También existen bibliotecas para la inicialización y manejo de esta tarjeta.

Si bien ambas tarjetas son soluciones válidas para las necesidades de nuestro proyecto, las características relativas al número de canales y sus características podrían limitar el desarrollo del mismo. Además suponen una solución con un coste muy elevado y obligan a utilizar un ordenador lo que entra en conflicto con el objetivo del trabajo.

4.5.2 Lego Mindstorms NXT

A pesar de tratarse de una conocida marca de juguetes, Lego ha conseguido crear, a un precio competitivo, una gama de robótica muy utilizada en la educación. Concretamente el kit de Lego Mindstorms incluye actuadores en forma de motores con encoder, reductora, y control por modulación de ancho de pulso muy similar a nuestro sistema. También incluye una variedad de sensores para aplicación en multitud de proyectos. Todo ello gobernado por un pequeño PLC, el NXT.

El bloque NXT incorpora todos los elementos necesarios en una sistema de adquisición de datos y presenta completa funcionalidad junto a los actuadores diseñados para el mismo. Utiliza tanto un lenguaje de programación propio a partir de bloques, como la posibilidad de su programación mediante el entorno RobotC en java o C.

Sin embargo, debido a la limitación en el número de entradas, a la limitación de memoria y procesador, al diseño cerrado del mismo y al elevado precio en comparación con otras soluciones descartaremos esta posibilidad.

Figura 48. Lego Mindstorms NXT.

4.5.2 Microchip PIC

Una alternativa a las tarjetas de adquisición de datos es el uso de microcontroladores que integren un sistema de adquisición con los periféricos necesarios y un procesador. Uno de los microcontroladores más utilizados ha sido el PIC de Microchip. Este incluye en un solo circuito integrado un microcontrolador completo de 8, 16 o 32 bits. Sin embargo, pese a su reducido precio, la programación y uso de este micro requiere de elementos adicionales como programadores y los kits de desarrollo y evaluación están muy enfocados a aplicaciones concretas, por lo que requiere en muchas ocasiones el diseño y elaboración de una placa para su uso en un proyecto concreto.

4.5.3 Arduino

En relación con los microcontroladores de bajo precio Arduino se ha posicionado en el mercado internacional como el número uno de los microcontroladores para proyectos de prototipado y con aplicaciones no comerciales. Cada dispositivo de esta marca está formado por un microcontrolador y una placa de desarrollo con las conexiones ya preparadas para cualquier tipo de uso que se le desee dar. Su sencillo entorno de programación y la inmensa comunidad que lo utiliza y lo respalda, ha conseguido que este sistema se convierta en el rey de los proyectos DIY (Do It Yourself) que ha promocionado el movimiento Maker.

4.5.4 ST STM32F4

Si bien la gran cantidad de bibliotecas y proyectos ya elaborados hacen de Arduino una solución interesante respecto a las demás, es ST y la arquitectura ARM la que dan verdadera potencia al mundo de los microcontroladores. Los microcontroladores basados en las familias Cortex-M aparecen en cualquier elemento comercial que utilicen un sistema embebido. Desde cafeteras hasta los frenos ABS de los coches de última generación. Además ST dispone de una gran variedad de kits de desarrollo para cualquier tipo de proyectos a precio reducido que compite con Arduino o PIC. Estos kits son fácilmente programables con los IDE como el Keil uVision completamente gratuitos que ST pone a disposición de desarrolladores en su página web.

En nuestro caso, por la gran potencia y posibilidades de desarrollo, así como por el creciente uso a nivel industrial y comercial de este, el microcontrolador en el que se desarrollará el control y que hará las veces de tarjeta de adquisición será un Cortex-M4 de ST, el kit de desarrollo y evaluación STM32F429-Discovery. La programación e implementación se desarrollarán en el apartado 5.

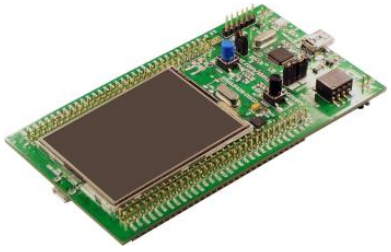


Figura 49. STM32F429-Discovery.

4.6 Elección de la acción de control

Existen diferentes formas de excitación de un motor de corriente continua para obtener el comportamiento deseado. Además, debido a las tensiones y corrientes elevadas, desde el punto de vista de la electrónica del microcontrolador, necesitamos una etapa de potencia que permita al microcontrolador operar de forma segura. En este punto estudiaremos varias posibilidades, y elegiremos una de ellas.

4.6.1 Conversión D/A. Acción de control directa bipolar

Aunque realicemos un control discreto en nuestro microcontrolador, no debemos perder de vista que el sistema a controlar consta de un proceso continuo. Por ello, es necesaria la conversión de la acción de control calculada mediante un convertidor D/A para poder aplicarla en nuestro sistema en cada instante. El propio microcontrolador que hemos elegido incorpora un convertidor D/A que podríamos utilizar para transmitir la salida a una etapa de potencia.

Puesto que necesitamos el giro del motor en ambos sentidos, necesitamos que la acción de control a la entrada del motor sea bipolar. Sin embargo, el microcontrolador solo puede proporcionar valores positivos de tensión entre 0 y $+V_{dd}$.

Una posible etapa de potencia podría consistir en un sencillo circuito con un amplificador operacional que alimentado por una fuente externa permita obtener una salida bipolar en función de la tensión de entrada.

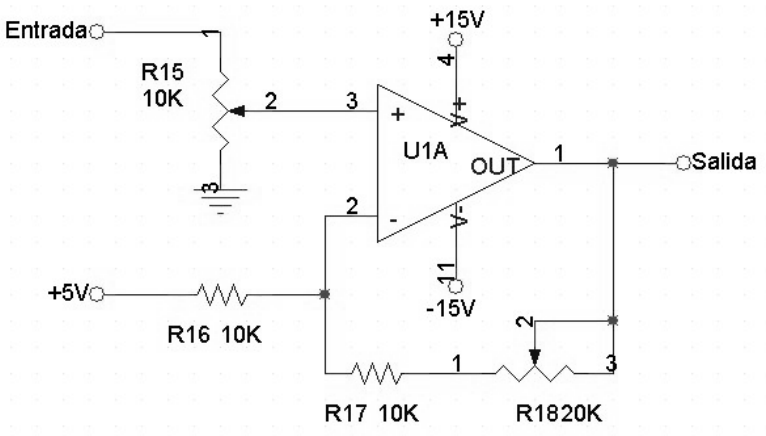


Figura 50. Ejemplo de etapa de potencia.

La construcción de este circuito presenta un inconveniente adicional en la elaboración del

proyecto, especialmente cuando existen soluciones ya implementadas de etapas de potencia, por tanto, pese a la posibilidad de control del motor mediante este método directo, se plantean otros bajo un punto de vista distinto.

4.6.2 Modulación de ancho de pulso (PWM)

Desde el punto de vista de la electrónica necesaria para accionar un motor de corriente continua, ya sea de imán permanente o con escobillas, existen multitud de soluciones: [Tal-76], [Jahns-90]. Aquellas basadas en la utilización de convertidores en los que sus interruptores electrónicos de potencia (transistores bipolares, y fundamentalmente, MOSFET) funcionan en modo conmutación son de las más eficientes. Esto es debido a que las pérdidas del convertidor en conmutación son mucho menores que cuando se trabaja en la zona activa (o en la zona ohmnica para los MOSFETs).

Este hecho es aún más importante si se trabaja a bajas velocidades y con un par elevado en la carga, ya que en estas condiciones, la fuerza contraelectromotriz (FCEM) del motor es muy baja (baja velocidad de giro) y la corriente de armadura es muy alta (alto par), con lo que la potencia puesta en juego es muy alta.

Como se acaba de indicar, estos convertidores utilizan transistores en modo conmutación, es decir que funcionan en modo corte y en modo saturación. El estado *on* se caracteriza porque la caída de tensión en el interruptor de potencia es muy baja. Puede darse el caso de que en este estado circule una corriente nula por el transistor, o puede darse el caso de que circule una corriente estando el transistor en saturación, lo que llamamos estado de conducción.

En el estado *off*, aunque la tensión en sus bornes sea alta, la corriente que circula por el transistor es muy baja (de microamperios o inferior), estando el interruptor en corte. De esta forma, en ambos casos la potencia disipada en dichos interruptores es muy baja (solo la debida al estado de conducción y la debida a la conmutación), lo que los hace especialmente interesantes en este tipo de aplicaciones donde se ponen en juego grandes potencias y no dejan de suponer una ventaja en el resto.

Existen diferentes estrategias de conmutación de los interruptores electrónicos de los convertidores. Una de las estrategias más sencillas es conmutar los transistores del convertidor a una frecuencia fija y modificar la duración de las fases en *on* y en *off*, es decir, su ciclo de trabajo (δ). Este es el tipo de modulación por anchura de pulso, o Pulse Width Modulation (PWM) en su traducción al inglés, cuyo diagrama de bloques puede verse en la siguiente figura.

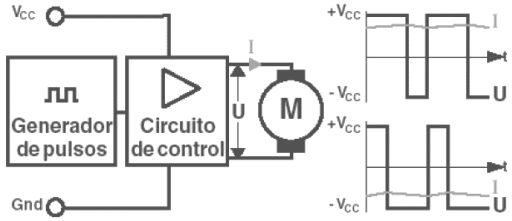


Figura 51: Diagrama de bloques de la modulación por ancho de pulso

Existen también otras configuraciones en las que la estrategia de conmutación es distinta, por ejemplo ya no sólo se varía la duración de las fases en *on* y en *off*, sino que también se varía la frecuencia de la señal modulada. Estos convertidores permiten un control con una variable adicional, proporcionando evidentes ventajas de diseño, e incluso permiten mantener un nivel de corriente del motor. Sin embargo, tienen el inconveniente de que al variar la frecuencia de modulación sobre un amplio rango se producen, en algunos casos, oscilaciones y ruidos audibles

muy molestos en el motor, debido a fenómenos de resonancias mecánicas que hacen que sean de utilización poco frecuente.

En este trabajo, no obstante, sólo se ha analizado el PWM con frecuencia de conmutación fija, ya que es el que se ha decidido implementado en la práctica, si bien existe la posibilidad de modificar la frecuencia de la señal modulada en cualquier momento como se indica en el apartado de la implementación sobre la programación del generador de la señal en nuestro microcontrolador.

A la vista de este estudio, la etapa de potencia que hemos decidido utilizar es el CI L298 y el control del motor se hará mediante modulación de ancho de pulso.Esta etapa de potencia permite recibir dos entradas PWM que se transmitirán a la salida, pudiendo controlar cada una un sentido de giro, existiendo la posibilidad de frenado brusco activando ambas entradas a la vez. Se describirá la implementación de estas señales en el microcontrolador en el apartado 5.

El esquema del control que tendremos sobre nuestro motor lo ilustra la siguiente figura.

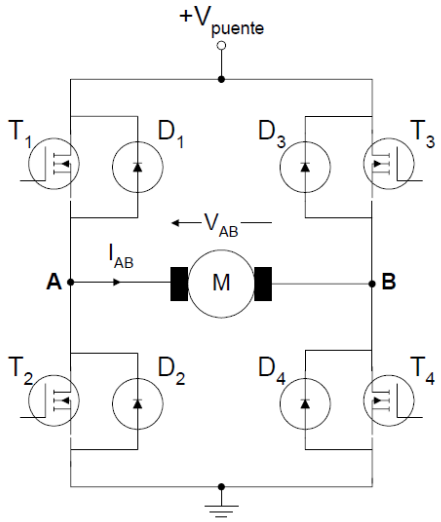


Figura 52: Etapa de potencia en puente de diodos

Donde, como podemos apreciar, se utiliza un puente de diodos gobernado por transistores (en nuestro caso serán transistores bipolares en lugar de MOSFETs). Este esquema ya se encuentra implementado en la placa junto al L298. Los terminales A y B, corresponden con la salida del puente al motor. El propio L298 permite cambiar la referencia entre ambos terminales para aplicar una señal positiva o negativa en función del sentido de giro deseado y que indicaremos mediante los pines 5 y 7, siempre que la entrada de enable (pin 6) esté a nivel alto (lo que siempre ocurrirá ya que la puentearemos). De esta forma si nuestra señal PWM generada en el microcontrolador llega al pin 7. Nuestro motor girará en un sentido, estableciéndose A o B una como referencia respecto a la otra. Sin embargo, si nuestra entrada llega al pin 5, el sentido de giro será el contrario, y la referencia se invertirá. La tercera opción de control será enviar una señal a ambos pines al mismo tiempo, de forma que si coinciden en un nivel alto, se bloqueará eléctricamente el eje del motor, lo que evitaremos en la medida de lo posible por ser notablemente perjudicial para nuestro proceso.

La siguiente figura ilustra la salida del puente de nuestra etapa de potencia con distintos ciclos de trabajo y el valor medio de la señal resultante que “notará” el motor.

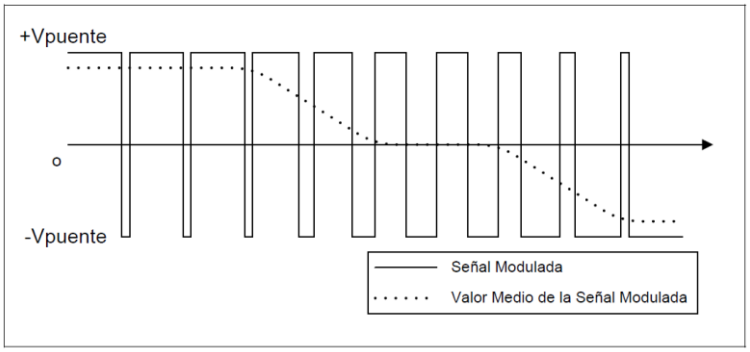


Figura 53: Señal de salida del puente de diodos

Si tomamos como referencia la hoja de características del anexo sobre el L298 [ANEXO A3.2], podemos ver el tiempo que tarda en conmutar la corriente de salida ante un cambio de tensión en la entrada. Podemos, por tanto, calcular el periodo mínimo de la señal PWM que generamos para respetar este límite si queremos una acción de control lo más eficaz posible. Podemos trabajar con seguridad hasta al menos 20 kHz, pero hemos de tener en cuenta que un aumento de la frecuencia implica un aumento del número de conmutaciones por unidad de tiempo y esto conlleva un aumento de las perdidas en el transistor.

Existe una segunda posibilidad a la hora de controlar nuestra etapa de potencia que consiste en gobernarla mediante dos entradas digitales en los pines 5 y 7, y una salida PWM que indique cuando se realiza la conmutación en el pin de enable (6). Sin embargo optaremos por la forma antes indicada.

4.7 Control del proceso

4.7.1 Controladores proporcionales, derivativos e integrales (PID)

El primer controlador a implementar por su simplicidad y por la facilidad a la hora de validar el proyecto y comprobar el seguimiento de trayectorias de nuestro generador en el sistema físico es el tipo PID. El PID incluye acción integral que asegura el seguimiento de la referencia si el proceso no tiene integrador e incluye una acción derivativa, lo que implica mejora de la respuesta en el transitorio y una mejora en el tiempo de respuesta, pero puede provocar que el sistema se vuelva inestable.

La acción más básica de control es la proporcional. Esta es, como indica su nombre, proporcional a la señal de error entre la salida del sistema y la referencia que se le especifica:

$$u(t) = k_p e(t)$$

Donde al parámetro *kp* se le denomina constante o ganancia proporcional. El diseño de la estrategia de control consiste únicamente en escoger el valor adecuado de dicho parámetro.

Por otro lado, si se dispone de una acción de control derivativa, la señal de control será proporcional a la derivada de la señal de error:

$$u(t) = k_d \frac{de(t)}{dt}$$

En este caso, el basarse en la tendencia (derivada) del error, permite "anticiparse" a dicho

error, por lo que la respuesta del sistema será, en general, más rápida.

La última acción básica de control es el control integral. Como no podía ser de otra forma, en este caso la acción de control es proporcional a la integral del error.

$$u(t) = k_i \int_0^t e(t)dt$$

Lo más destacable de este tipo de acción de control es que *u(t)* únicamente tendrá un valor constante cuando la señal de error *e(t)* sea igual a cero. Por ello esta acción de control permitirá eliminar, en el régimen estable, los errores entre la salida y la referencia del sistema a controlar.

A la hora de establecer el control por computador mediante estas acciones de control se debe tener en cuenta que no se suelen aplicar las acciones de control de forma individual, sino que por el contrario el controlador suele ser una combinación de estas acciones básicas. Además, se debe implementar los controladores en un algoritmo o programa de control, por lo que se necesita la versión discreta de los controladores.

-Control proporcional (P): La implementación del algoritmo de control en este caso es muy simple puesto que lo único que hay que hacer es obtener la ecuación en diferencias de la expresión anterior:

$$u(k) = k_p e(k)$$

Ecuación 31. Ecuación en diferencias del control proporcional.

Donde *u(k)* es la acción de control a aplicar en el instante k, que depende de la ganancia proporcional del controlador y del error entre la referencia y la salida en dicho instante.

-Control Proporcional-Derivativo (PI): El controlador combina la acción proporcional y derivativa:

$$u(t) = K_{PD}\left(e(t) + T_d \frac{de(t)}{dt}\right)$$

Para poder implementarlo en el algoritmo de control se debe obtener el controlador discreto equivalente, y para ello se puede aproximar la función derivada del error por el cociente incremental:

$$\frac{de(t)}{dt} \cong \frac{e(k) - e(k-1)}{T}$$

Siendo T el periodo de muestreo utilizado. Desarrollando la expresión del controlador, la versión discreta de éste será la siguiente:

$$G_{R_{PD}}(z) = \frac{q_0 z + q_1}{z}$$

Donde:

$$q_0 = K_{PD}\frac{T - T_d}{T}$$

$$q_1 = -K_{PD}\frac{T_d}{T}$$

De esta forma, la acción de control de este controlador será:

$$u(k) = q_0 e(k) + q_1 e(k-1)$$

Ecuación 32. Ecuación en diferencias del control proporcional-derivativo

Lo que indica que la acción de control en el instante *k* depende no sólo del error entre la referencia y la salida en el instante *k*, sino también del error en el instante anterior.

-Control Proporcional-Integral (PI): En este controlador, la acción de control tiene la siguiente expresión:

$$u(t) = K_{PI}\left(e(t) + \frac{1}{T_i}\int_0^t e dt\right)$$

Al igual que en el caso anterior, se debe obtener su equivalente discreto para poder implementarlo con un algoritmo de control. En este caso se tienen varias opciones para aproximar la integral del error. Una de ellas es considerar que la integral es una suma de áreas:

$$\int_0^t e dt \cong \sum_{i=0}^{k-1} T e(i)$$

Con esta aproximación, la expresión del controlador proporcional-integral será la siguiente:

$$G_{R\,PI}(z) = \frac{q_0 z + q_1}{z - 1}$$

Donde:

$$q_0 = K_{PI}$$

$$q_1 = K_{PI}\frac{T - T_i}{T}$$

La acción de control del controlador PI será:

$$u(k) = q_0 e(k) + q_1 e(k-1) + u(k-1)$$

Ecuación 33. Ecuación en diferencias del control proporcional-integral

Como se puede apreciar, la acción de control del PI en el instante *k* depende de los valores del error en el instante *k* y *k-1*, así como de la acción de control aplicada en el instante anterior.

-Control Proporcional-Integral-Derivativo (PID): En este controlador, la acción de control viene dada por las aportaciones proporcional, derivativa e integral:

$$u(t) = K_{PID}\left(e(t) + \frac{1}{T_i}\int_0^t e dt + T_d\frac{de(t)}{dt}\right)$$

Considerando las aproximaciones de la derivada y de la integral planteadas anteriormente, la expresión del regulador PID será la siguiente:

$$G_{R_{PID}}(z) = \frac{q_0 z^2 + q_1 z + q_2}{z(z-1)}$$

Donde:

$$q_0 = K_{PID}\frac{T + T_d}{T}$$

$$q_1 = K_{PID}\left(-1 + \frac{T}{T_i} - \frac{2T_d}{T}\right)$$

$$q_2 = K_{PID}\frac{T_d}{T}$$

La acción de control del PID será:

$$u(k) = q_0 e(k) + q_1 e(k-1) + q_2 e(k+2) + u(k-1)$$

Ecuación 33. Ecuación en diferencias del control proporcional-integral-derivador

La acción de control del PID en el instante *k* depende de los valores del error en el instante *k*, *k-1* y *k-2*, así como de la acción de control aplicada en el instante anterior.

Para la validación del proyecto se llevará a cabo la implementación de estos reguladores en su versión discreta, como forma de control del seguimiento de nuestros patrones de movimiento, debido a la robustez que presentan, la facilidad para regular sus parámetros y la comodidad de diseño, mediante métodos experimentales. Se desarrollará en la descripción detallada de la solución adoptada el diseño de estos controladores.

4.7.2 Espacio de estados. Control por realimentación del estado observado

Una alternativa de control analógico al PID sería plantear nuestro modelo de motor desde la siguiente perspectiva, partiendo del modelo matemático de posición, en la que cada bloque se corresponde con la función de transferencia resultante de la ecuación eléctrica, la ecuación mecánica y el integrador de la posición respectivamente:

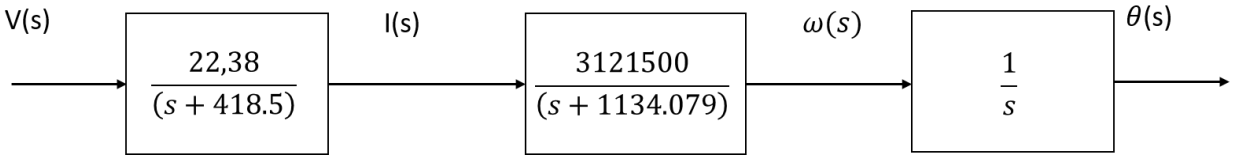


Figura 54. Modelo del espacio de estados del motor de corriente continua

De esta manera, podríamos definir 3 estados para nuestro sistema:

$$x = [I \quad \omega \quad \theta]^T$$

Y podríamos expresar nuestro sistema en el espacio de estados:

Para la ecuación eléctrica:

$$I(s) = \frac{22.38}{s + 418.5} V(s)$$

$$(s + 418.5) I(s) = 22.38\, V(s)$$

$$\big(sI(s)\big) = 22.38\, V(s) - 418.5 I(s)$$

$$\frac{dI(t)}{dt} = 22.38\, V(t) - 418.5 I(t)$$

Para la ecuación mecánica:

$$\omega(s) = \frac{3121500}{s + 1134.079} I(s)$$

$$(s + 1134.079)\omega(s) = 3121500 I(s)$$

$$\big(s\omega(s)\big) = 3121500\, I(s) - 1134.079\omega(s)$$

$$\frac{d\omega(t)}{dt} = 3121500\, I(t) - 1134.079\omega(i)$$

Y para el integrador:

$$\theta(s) = \frac{1}{s}\omega(s)$$

$$s\theta(s) = \omega(s)$$

$$\frac{d\theta(t)}{dt} = \omega(t)$$

Resultando:

$$\dot{x} = \begin{bmatrix} -418.5 & 0 & 0 \\ 3121500 & -1134.079 & 0 \\ 0 & 1 & 0 \end{bmatrix} \cdot x + \begin{bmatrix} 22.38 \\ 0 \\ 0 \end{bmatrix} \cdot V = A \cdot x + B \cdot V$$

Si se define como criterio de diseño:

$$t_{e(98\%)} = 0.2s$$

$$\delta = 10\%$$

Se tiene como polos a asignar:

$$\sigma = \frac{4}{t_e} = 20$$

$$\omega_p = \frac{\pi\sigma}{-ln\delta} = 27.28$$

$$p = \left[-\sigma + \omega_p j, -\sigma - \omega_p j, -1.5\sigma\right]$$

Cuya ecuación característica será:

$$\big(s - p(1)\big)\big(s - p(2)\big)\big(s - p(3)\big) = ((s - \sigma)^2 + {\omega_p}^2)(s - 1.5\sigma)$$

$$s^3 - 70s^2 + 2345s - 34343.827$$

Con el modelo del espacio de estados en mente:

$$\begin{cases} \dot{x} = A \cdot x + B \cdot V & V = F \cdot r - K \cdot x \\ y = \theta & K = \begin{bmatrix} k_1 & k_2 & k_3 \end{bmatrix} \end{cases}$$
$$\dot{x} = (A - B \cdot K) \cdot x + B \cdot F \cdot r$$

Ecuación 34. Formulación del espacio de estados

Podemos asignar los polos mediante la ecuación:

$$|sI - (A - B \cdot K)| = s^3 - 70s^2 + 2345s - 34343.827$$

Despejando K:

$$\begin{vmatrix} s + 418.5 - 22.38k_1 & 22.38k_2 & 22.38k_3 \\ -3121500 & s + 1134.079 & 0 \\ 0 & -1 & s \end{vmatrix}$$
$$= s^3 + (418.5 - 22.38k_1 + 1134.079)s^2 + \big((1134.079 \cdot 418.5) - (1134.079 \cdot 22.38k_1) + (3121500 \cdot 22.38k_2)\big)s + (-3121500 \cdot 22.38k_3)$$

Agrupando por términos con el mismo exponente:

$$(418.5 - 22.38k_1 + 1134.079) = 70$$
$$(1134.079 \cdot 418.5) - (1134.079 \cdot 22.38k_1) + (3121500 \cdot 22.38k_2) = 2345$$
$$-3121500 \cdot 22.38k_3 = 34343.827$$

Siendo el resultado del sistema:

$$K = [66.25 \quad 0.017 \quad 0]$$

La función de transferencia en bucle cerrado sería:

$$G_{BC}(s) = \lceil 0 \quad 0 \quad 1 \rceil \cdot (sI - A + B \cdot K)^{-1} \cdot B = \frac{6994047.619}{s^3 - 70s^2 + 2345s - 34343.827}$$

Obteniendo F como:

$$G_{BC}(0) = \frac{6994047.619}{34343.827} = 203.64 \rightarrow F = G_{BC}(0)^{-1} = 4.9\ 10^{-3}$$

Y con el siguiente esquema de Simulink, habiendo inicializado las variables del sistema en el script de Matlab del anexo [ANEXO A1.0], podemos simular la respuesta:

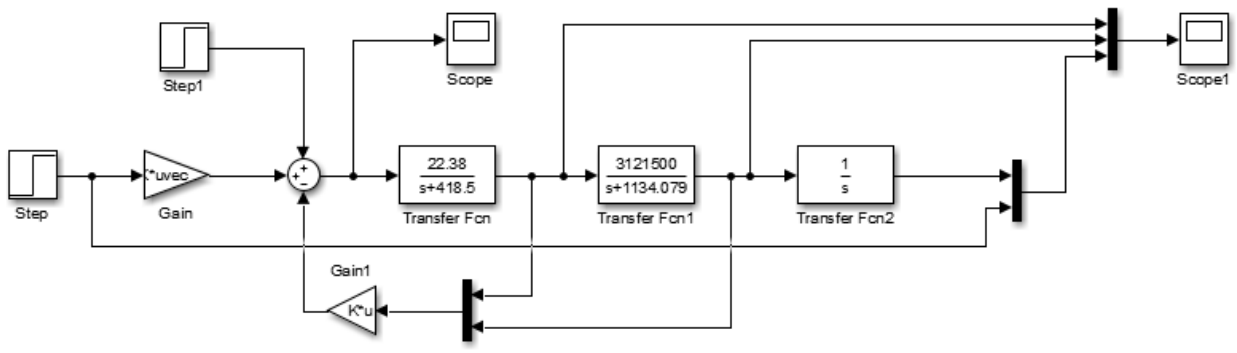


Figura 55. Esquema para la simulación del control por realimentación del estado

Obteniendo como resultado:

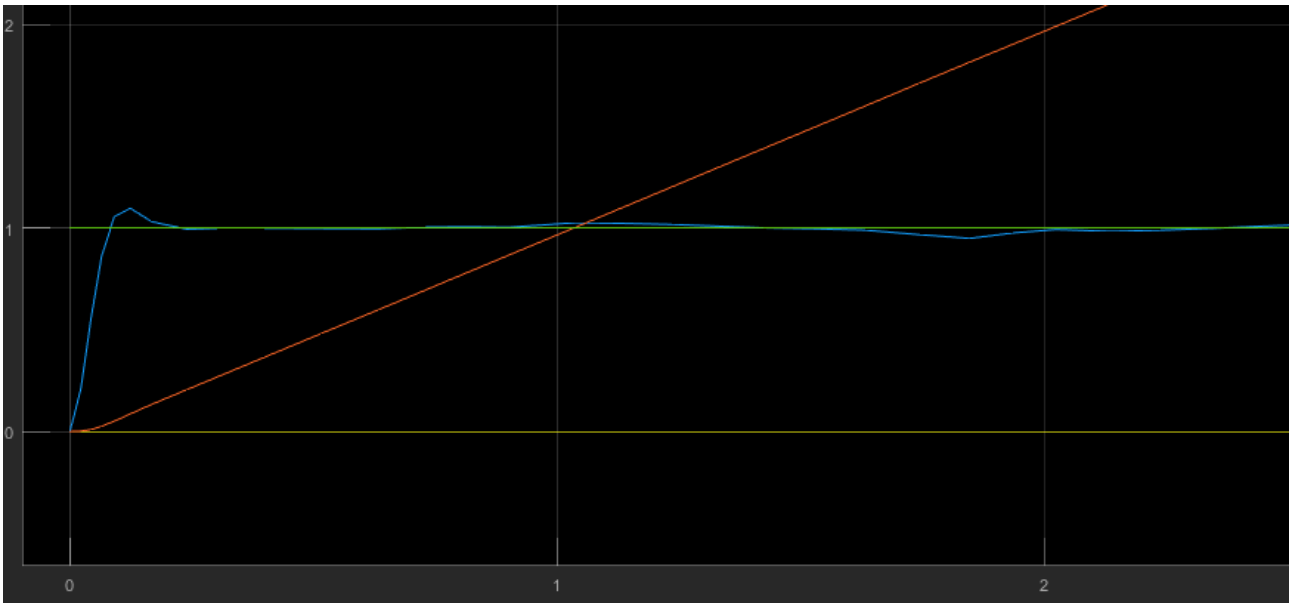


Figura 56. Respuesta ante escalón del control por realimentación del estado

Donde, como puede apreciarse, se consiguen los resultados deseados, controlando la posición del eje mediante una velocidad respuesta a un escalón de entrada, ante una velocidad de 1º/s conseguimos una posición del eje de 1º en el tiempo que cabría esperar. El tiempo de establecimiento y la sobreoscilación cumplen los criterios de diseño.

Para la velocidad, el sistema carece del integrador. Podemos definir el nuevo modelo del espacio de estados como un subsistema del anterior. Este control, no obstante, no elimina perturbaciones. Podría realizarse un control con acción integral para solucionarlo. En cualquier caso, la implementación en un microcontrolador requiere un enfoque diferente y por la limitación del proyecto y los objetivos del mismo no se optará por este control para la solución desarrollada.

4.7.3 Control por asignación de polos

La capacidad de control de los controladores analógicos y la comodidad en su diseño es evidente, sin embargo, la realización de los mismos en un sistema digital supone una serie de dificultades que ya se han explicado. Una alternativa eficaz es el diseño e implementación de reguladores algebraicos discretos.

El control más directo ante un proceso conocido consiste en obtener un regulador, en este caso discreto, que asigne las raíces solución de la ecuación característica en bucle cerrado según el diseño.

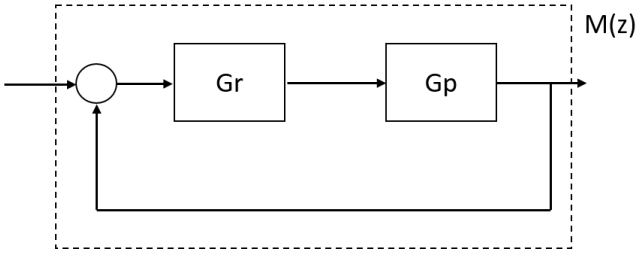


Figura 57. Diagrama de bloques del bucle cerrado de control

$$M(z) = \frac{G_r(z)G_p(z)}{1 + G_r(z)G_p(z)}$$

Ecuación 35: Función de transferencia en bucle cerrado del sistema

Si suponemos:

$$G_p(z) = \frac{B(z)}{A(z)} = \frac{b_m z^m + b_{m-1} z^{m-1} + \cdots + b_1 z + b_0}{z^n + a_{n-1} z^{n-1} + \cdots + a_1 z + a_0}$$

$$G_r(z) = \frac{Q(z)}{P(z)} = \frac{q_u z^u + q_{u-1} z^{u-1} + \cdots + q_1 z + q_0}{z^v + p_{v-1} z^{v-1} + \cdots + p_1 z + p_0}$$

Ecuación 36: Funciones de transferencia de proceso y regulador

Siendo p_i los polos a asignar que cumplen las especificaciones, puede plantearse la siguiente ecuación, conocida como ecuación diofántica, como solución para el diseño del controlador:

$$A(z)P(z) + B(z)Q(z) = \prod_i (z - p_i)$$

Ecuación 37: Ecuación diofántica

El diseño del controlador consistirá en obtener P(z) y Q(z) a partir de la relación de la ecuación 37.

Para ello se deben tener en cuenta algunos aspectos:

- El controlador debe ser realizable.

- A(z) y B(z) deben ser primos entre sí.

- El número de coeficientes de la ecuación característica debe ser igual al de parámetros del controlador.

Si se cumplen estos aspectos el controlador tendrá solución única. En definitiva, existe solución única si, en la ecuación 36:

$$n + v = u + v + 1$$

Entonces:

$$u = n - 1 \text{ y } v \geq u$$

Si queremos evitar retardos en el sistema, se tomará el mismo valor de v y u:

$$u = v \rightarrow \text{nº polos a asignar: } orden\ Ec\ Diof. = n + v = 2n - 1$$

Existe la posibilidad, mediante este controlador, de cancelar el efecto de los ceros del proceso asignando un polo al lugar exacto del cero. Sin embargo, no deben cancelarse ceros inestables para evitar comportamiento inestable en bucle cerrado por las discrepancias entre el modelo y el proceso real.

Si se desea eliminar el error en régimen permanente para un proceso sin integrador se puede considerar éste dentro del proceso, y calcular el controlador normalmente, incluyendo este integrador posteriormente en el regulador.

Partiendo del modelo discreto obtenido en el apartado de modelado a partir de la respuesta ante entrada en escalón, se va a diseñar un controlador por asignación de polos que anule el error de posición, cancele el cero del proceso y consiga las condiciones de diseño que se han utilizado a lo largo del trabajo.

$$G_p(z) = \frac{0.0048z + 0.0046}{z^2 - 1.8752z + 0.8752}$$

$$G_p(z) = \frac{0.0048(z + 0.96)}{(z - 1)(z - 0.8752)}$$

Como el proceso ya incluye integrador no hace falta incluirlo para alcanzar el error nulo de posición. Para los criterios de diseño los polos son:

$$te = \frac{4}{\sigma} \leq 0.1s \rightarrow \sigma = \frac{4}{0.1} = 40$$

$$\delta(\%) \leq 5\% \rightarrow \delta = e^{\frac{-\sigma\pi}{wp}} \rightarrow wp = 41.95$$

$$s = -40 \pm 41.95j$$

Se discretizan para el periodo que se ha estado utilizando: T=0.01s

$$z = e^{sT} = e^{-0.4 \pm 0.4195j} = e^{-0.4}(\cos(0.4195) \pm jsen(0.4195))$$

$$z = 0.6122 \pm 0.273j$$

Para:

$$\mathrm{u} = \mathrm{v} = \mathrm{n} - 1 = 1$$

Se tiene:

$$G_r(z) = \frac{Q(z)}{P(z)} = \frac{q_1 z + q_0}{z + p_0}$$

El número de polos a asignar es:

$$2\mathrm{n} - 1 = 3$$

Estos serán los dos de diseño y el del proceso. Así se plantea la ecuación diofántica:

$$(z - 1)(z - 0.8752)(z + p_0) + 0.0048(z + 0.96)(q_1 z + q_0) = [(z - 0.6122)^2 + 0.273^2](z + 0.96)$$

Si aplicamos la propiedad que dice que el denominador del regulador incluye todos los ceros del proceso que se puedan cancelar:

$$(z + p_0) = (z + 0.96) \rightarrow p_0 = 0.96$$

Y para el resto:

$$z^2 - 0.8752z - z + 0.8752 + 0.0048q_1 z + q_0 0.0048 = z^2 - 1.2244z + 0.4485$$

Agrupando por exponentes:

$$-0.8752 - 1 + 0.0048q_1 = -1.2244 \rightarrow q_1 = 135.58$$

$$0.8752 + q_0 0.0048 = 0.4485 \rightarrow q_0 = -88.89$$

Por lo que:

$$G_r(z) = \frac{135.58z - 88.89}{z + 0.96}$$

Y en el bucle cerrado:

$$M(z) = \frac{\frac{0.0048(135.58z - 88.89)}{(z-1)(z-0.8752)}}{\frac{(z-1)(z-0.8752) + 0.0048(135.58z - 88.89)}{(z-1)(z-0.8752)}}$$

$$M(z) = \frac{(0.65z - 0.4265)}{(z-1)(z-0.8752) + (0.65z - 0.4265)}$$

Ecuación 38. Función de transferencia en bucle cerrado del sistema tras el control diseñado

Podemos simular la respuesta del sistema con el regulador diseñado mediante el siguiente esquema de Simulink:

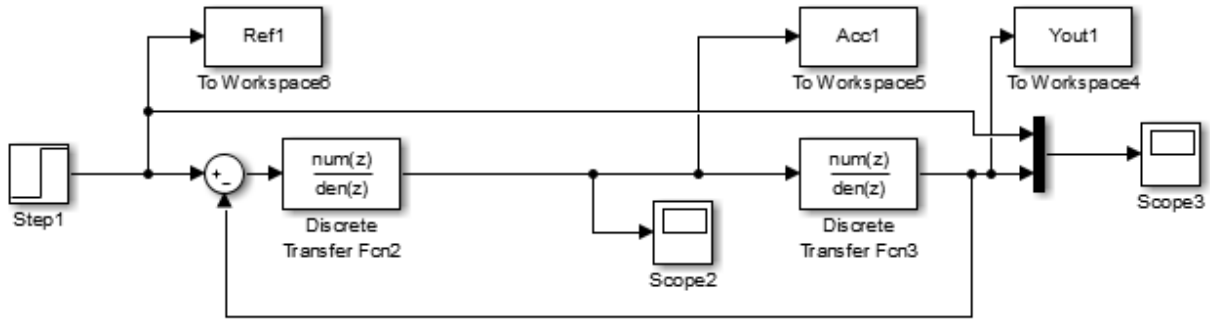


Figura 58. Esquema para la simulación del control de asignación de polos

Donde se obtiene la siguiente respuesta ante escalón:

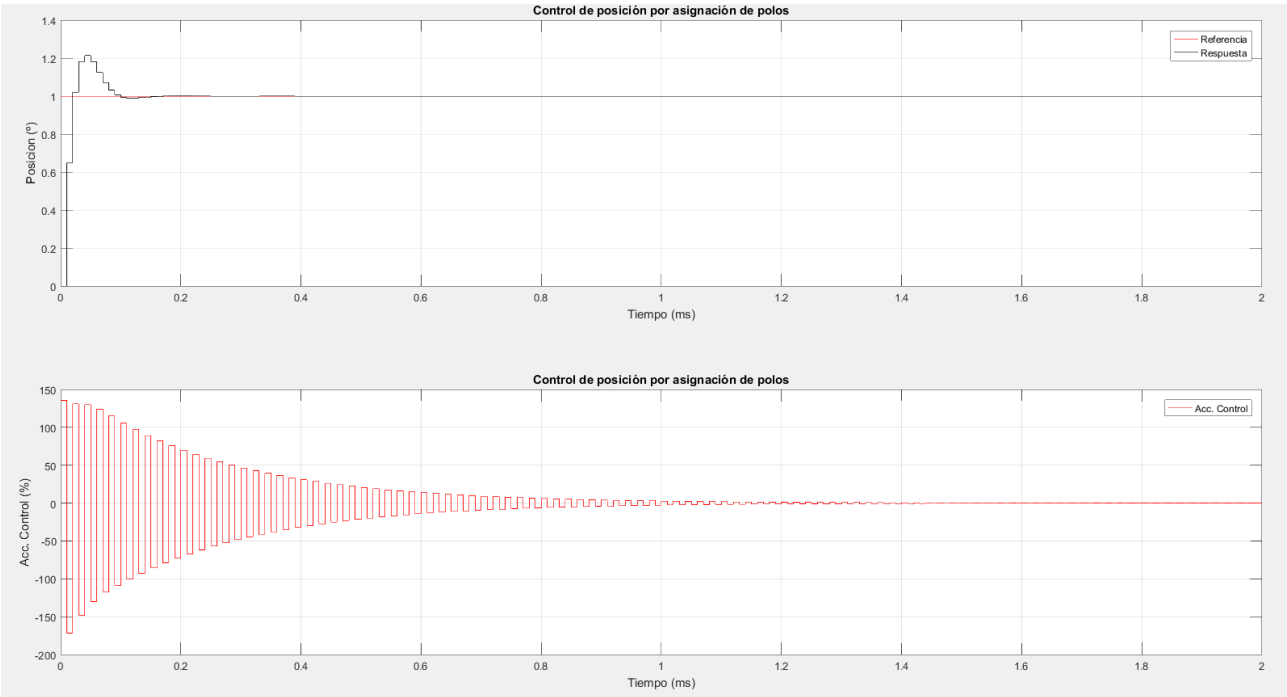


Figura 59. Respuesta ante escalón del control de asignación de polos en bucle cerrado

Como se puede apreciar, se alcanza la referencia en el tiempo especificado y con la sobreoscilación deseada, pero la acción de control es mayor de la que nuestro sistema puede

proporcionar. Esto es debido a que nuestras condiciones de diseño son muy restrictivas. Al relajar las especificaciones se tendría una acción de control de menor intensidad.

Con una sencilla implementación en Matlab, conociendo los valores de referencia de posición para un perfil de movimiento de los implementados, podemos obtener el máximo valor de referencia que se utilizará en un determinado instante para conseguir el movimiento deseado:

```
posold=0
for i=1:length(pos)
    dif(i)=pos(i)-posold;
    posold=pos(i)
end
max(dif)
```

Para el caso de nuestro sistema de ensayos la velocidad máxima que podemos obtener del motor experimentalmente es de 800º/s al 100% del ciclo de trabajo dentro del punto de funcionamiento en el que queremos controlar el motor. Además la distancia máxima lineal que podremos desplazar nuestro carro en el eje implementado es de 270mm, lo que equivale para nuestra configuración de trasmisión a 3750º. Así podemos definir los parámetros que limitan nuestro movimiento y obtener, a partir de ellos, las referencias que guiarían nuestro movimiento. Para estos valores el mayor valor de referencia que se aplicará con el perfil de movimiento trapezoidal es de un par de grados para la posición.

Podemos, por tanto, referirnos a este valor como el caso más desfavorable para comprobar el desempeño que realizan los controles diseñados. Así ante entrada de escalón de valor 2, el control diseñado se comporta así:

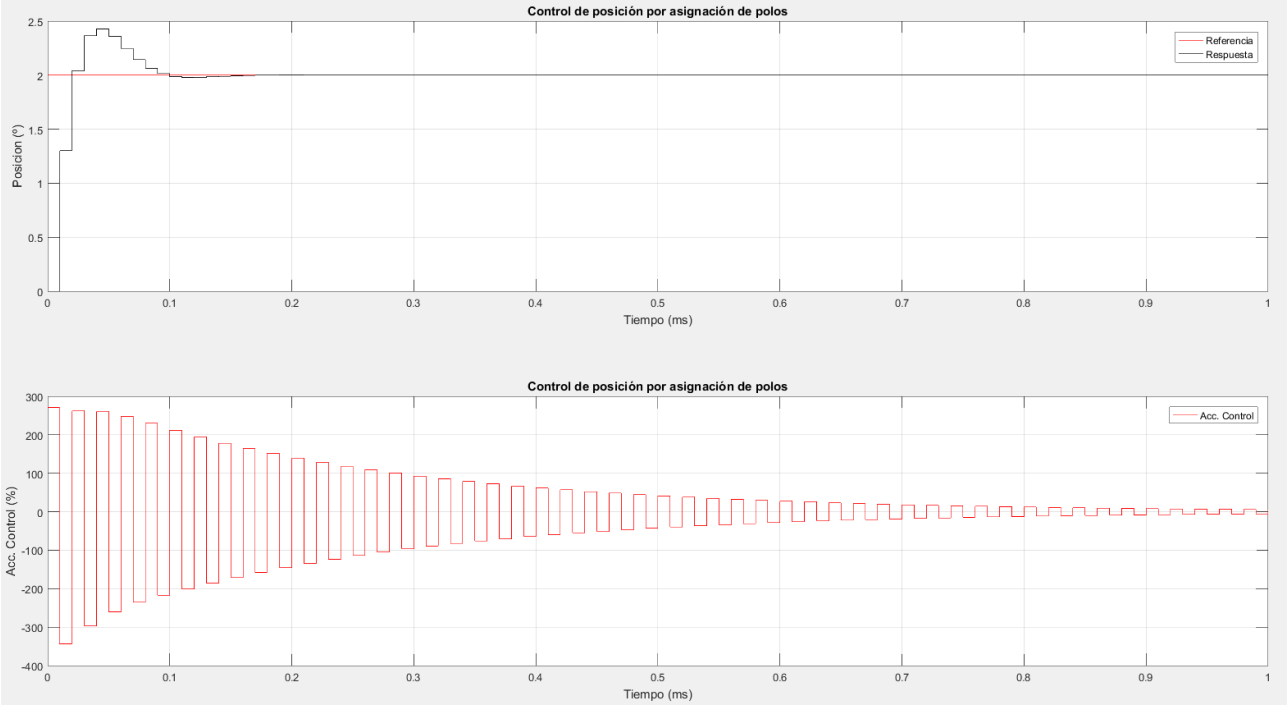


Figura 60. Respuesta ante escalón de valor 2 del control de asignación de polos en bucle cerrado

De forma que para las especificaciones de diseño que se han utilizado en la obtención de este regulador la acción de control escapa al rango que podemos aplicar. Además aparecen oscilaciones ocultas en la acción de control ya que, pese a haber alcanzado la referencia, la respuesta permanece oscilando. Si bien las condiciones de diseño se alcanzan, salvo por la sobreoscilación, la acción de control es muy superior a la que puede proporcionar nuestro sistema.

El diseño del control de velocidad seguiría el mismo planteamiento y desarrollo pero sabiendo que debe incluirse un integrador en el proceso para el diseño si se quiere conseguir error de posición nulo.

En el siguiente apartado se estudia otro tipo de controladores que permite evitar el fenómeno de las oscilaciones ocultas.

4.7.4 Control por cancelación. Control en tiempo finito y tiempo mínimo

Un controlador por cancelación es aquél que sustituye la dinámica del proceso por otra deseada dada por la función de transferencia en bucle cerrado M(z).

Si despejamos Gr(z) de la ecuación 35:

$$M(z) + M(z)G_r(z)G_p(z) = G_r(z)G_p(z)$$

$$M(z) + M(z)G_r(z)G_p(z) = G_r(z)G_p(z) - M(z)G_r(z)G_p(z) = (1 - M(z))G_r(z)G_p(z)$$

$$G_r(z) = \frac{1}{G_p(z)}\frac{M(z)}{1 - M(z)}$$

De esta forma, si calculamos el controlador en bucle cerrado que nos da la M(z) deseada podemos cancelar el proceso tal y como se aprecia en la función de transferencia en bucle abierto:

$$G_r(z)G_p(z) = \frac{1}{\cancel{G_p(z)}}\frac{M(z)}{1 - M(z)}\cancel{G_p(z)}$$

Existen unas condiciones necesarias para el cálculo de estos controladores:

- Se trabaja en z < 0, es decir, con exponentes de z negativos.

- El controlador debe ser realizable. Esto será si en la ecuación Nº2:

$$v \geq u$$

Normalmente se toma v=u para evitar retrasos.

- No deben cancelarse polos o ceros del proceso inestables (>1). Esto es debido a las diferencias entre el modelo matemático y el sistema real introducidas por la cuantificación de los sistemas discretos. Si:

$$G_p(z) = \frac{B(z)}{A(z)}$$

Es nuestro modelo, y:

$$\widetilde{G_p(z)} = \frac{\widetilde{B(z)}}{\widetilde{A(z)}}$$

Es el sistema real. Al aplicar el control al proceso:

$$G_r(z)\widetilde{G_p(z)} = \frac{A(z)}{B(z)}\frac{M(z)}{1 - M(z)}\frac{\widetilde{B(z)}}{\widetilde{A(z)}}$$

Ecuación 39. Ecuación de cancelación en bucle abierto del proceso

Si nuestro modelo no es lo suficientemente preciso estaremos intentando cancelar ceros inestables con polos inestables muy cercanos pero no lo bastante como para efectuar la cancelación.

Para evitarlo se incluyen los polos y ceros inestables del modelo, que es conocido, en la función de transferencia de bucle cerrado:

$$M(z) = \left[\sum_{i=1}^{q}(1 - b_i z^{-1})\right] M_2(z^{-1})\ con\ (1 - b_i z^{-1}) ceros\ de\ fase\ no\ minima\ del\ modelo$$

$$1 - M(z) = \left[\sum_{j=1}^{p}(1 - a_i z^{-1})\right] M_1(z^{-1})\ con\ (1 - a_i z^{-1}) polos\ inestables\ del\ modelo$$

- Si el la función de transferencia del proceso es estrictamente propia, es decir, el proceso tiene retardo, la función de transferencia de bucle cerrado M(z) debe tener al menos ese retardo.

Un tipo de controladores de cancelación son los Controladores de Tiempo Mínimo, que son aquéllos que hacen que la salida del bucle de control (y por tanto la salida del proceso) presente un error en régimen permanente nulo en el menor tiempo posible, dicho de otro modo, en el menor número posible de instantes de muestreo. Esto no asegura que el proceso continuo no evolucione entre los instantes de muestreo a pesar de tener error nulo, es decir, sólo se verifica que el error en régimen permanente es nulo en los instantes de muestreo. En consecuencia, la salida del proceso puede presentar las llamadas oscilaciones ocultas, que pueden no percibirse en tiempo discreto, a menos que analicemos la acción de control que no sería estable cuando debería serlo.

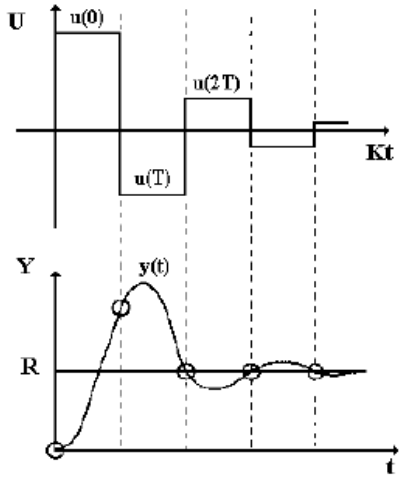


Figura 61. Oscilaciones ocultas

Matemáticamente los controladores de tiempo mínimo responden a la siguiente ecuación:

$$E(z^{-1}) = [1 - M(z^{-1})]R(z^{-1})$$

Ecuación 40. Ecuación de diseño de un control de tiempo mínimo

De manera que si $E(z^{-1})$ es un polinomio, es decir, no tiene denominador, se asegura un valor de error nulo a partir de L siendo:

$$E(z^{-1}) = \sum_{k=0}^{L} e(kT) z^{-k}$$

Si la referencia $R(z^{-1})$ tiene denominador, $[1 - M(z^{-1})]$ debe contenerlo para que $E(z^{-1})$

tenga un número finito de términos.

Los controladores de tiempo finito son aquellos que permiten eliminar el error en régimen permanente en el seguimiento de una determinada señal de referencia en el menor tiempo posible sin oscilaciones ocultas. Los controladores de tiempo finito son, también, controladores de cancelación por lo que su expresión obedecerá a la siguiente fórmula:

$$U(z^{-1}) = G_r(z^{-1})E(z^{-1}) = G_r(z^{-1})\frac{M(z^{-1})}{1-M(z^{-1})}[1-M(z^{-1})]R(z^{-1}) = \frac{A(z^{-1})}{B(z^{-1})}M(z^{-1})R(z^{-1})$$

Ecuación 41. Ecuación de diseño de un control de tiempo finito

Para conseguir una acción de control finita $U(z^{-1})$ no debe tener denominador, por lo que se incluirán todos los ceros del proceso en $M(z^{-1})$:

$$M(z^{-1}) = B(z^{-1})\,M_2(z^{-1})$$

Sin embargo, no podemos incluir en $M(z^{-1})$ a la referencia.

$$R(z^{-1}) = \frac{R_1(z^{-1})}{(1-z^{-1})^{m+1}}$$

De manera que la existencia de oscilaciones ocultas dependerá tanto del proceso como de la referencia.

Ante referencia escalón, un sistema de tipo 1 no presentará oscilaciones ocultas ya que m+1=1 y se anulará con el integrador del sistema. Pero ante referencia rampa, el mismo sistema sí tendrá oscilaciones ocultas.

Ante entrada escalón, un sistema de tipo 0 que no tiene integradores presentará una acción de control de valor fijo tras un número determinado de instantes:

$$U(z^{-1}) = A(z^{-1})M_2(z^{-1})\frac{1}{1-z^{-1}} = \frac{u_0 + u_1 z^{-1} + \cdots + u_k z^{-k}}{1-z^{-1}}$$

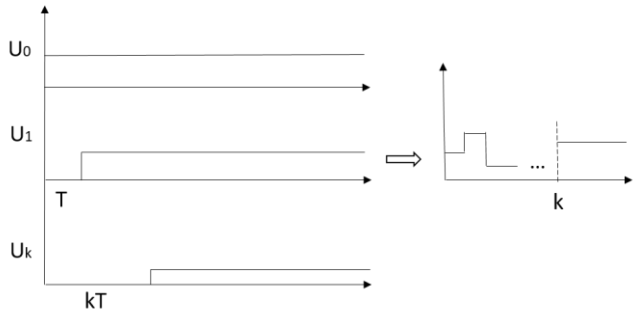


Figura 62. Entrada finita de valor constante para referencia escalón de un sistema de tipo 0

Como ya se ha indicado, estos controladores presentan acciones de control muy fuertes, por ello se debe tener en cuenta a la hora de diseñarlos que han de conseguirse las especificaciones pero manteniendo las acciones de control dentro de los rangos de los actuadores. Si restringimos el valor de las acciones de control, estamos introduciendo más incógnitas y más ecuaciones a nuestro sistema y por lo tanto más instantes antes de alcanzar la referencia.

Una vez definidos estos reguladores, se va a diseñar un regulador de tiempo finito para la posición y velocidad de nuestro modelo, de manera que anule el error de posición en el menor tiempo posible sin oscilaciones ocultas.

Partimos del modelo de posición discreto obtenido en el apartado de modelado no

paramétrico:

$$G_p(z) = \frac{0.0048z + 0.0046}{z^2 - 1.8752z + 0.8752}$$

$$G_p(z) = \frac{0.0048(z + 0.96)}{(z - 1)(z - 0.8752)}$$

Pasamos a z negativas:

$$G_p(z^{-1}) = \frac{0.0048z^{-1}(1 + 0.96z^{-1})}{(1 - z^{-1})(1 - 0.8752z^{-1})}$$

Analizamos las características del proceso:

- retardo: d=1
- ceros: $(1 + 0.96z^{-1})$ q=1
- referencia: escalón →m+1=1
- integradores del proceso: $\nu' = 1$
- Polos inestables: p=0

Por tanto:

$$M(z^{-1}) = z^{-1}(1 + 0.96z^{-1})M_2(z^{-1})$$

$$1 - M(z^{-1}) = (1 - z^{-1})^{\max(1,1)}M_1(z^{-1})$$

Y el número de instantes en el que alcanzaríamos la referencia sería:

$$orden\big(M(z^{-1})\big) = d + q + \max(m + 1, \nu') + p - 1 = 1 + 1 + 1 + 0 - 1 = 2$$

$$M_2(z) = d_0$$

$$M_1(z) = 1 + c_1 z^{-1}$$

Donde:

$$1 - (z^{-1}(1 + 0.96z^{-1})(d_0)) = (1 - z^{-1})(1 + c_1 z^{-1})$$

$$1 - \big((z^{-1} + 0.96z^{-2})(d_0)\big) = 1 + c_1 z^{-1} - z^{-1} - c_1 z^{-2}$$

$$1 - d_0 z^{-1} - 0.96 d_0 z^{-2} = 1 + c_1 z^{-1} - z^{-1} - c_1 z^{-2}$$

Agrupando por términos del mismo orden:

$$-0.96 d_0 = -c_1$$

$$-d_0 = c_1 - 1$$

De donde se obtiene:

$$d_0 = -c_1 + 1 \rightarrow d_0 = -0.4898 + 1 = 0.5102$$

$$-0.96(-c_1 + 1) = 0.96c_1 - 0.96 = -c_1 \rightarrow\ c_1 = \frac{0.96}{1.96} = 0.4898$$

Por lo que:

$$G_r(z^{-1}) = \frac{1}{G_p(z^{-1})} \frac{M(z^{-1})}{1 - M(z^{-1})}$$

$$G_r(z^{-1}) = \frac{(1-z^{-1})(1-0.8752z^{-1})}{0.0048z^{-1}(1+0.96z^{-1})}\frac{z^{-1}(1+0.96z^{-1})0.5102}{(1-z^{-1})(1+0.4898z^{-1})}$$

$$G_r(z^{-1}) = \frac{0.5102(1-0.8752z^{-1})}{0.0048\,(1+0.4898z^{-1})}$$

$$G_r(z) = \frac{0.5102(z-0.8752)}{0.0048\,(z+0.4898)}$$

Ecuación 42. Controlador de tiempo finito diseñado

Podemos simular este regulador mediante el siguiente esquema de Simulink:

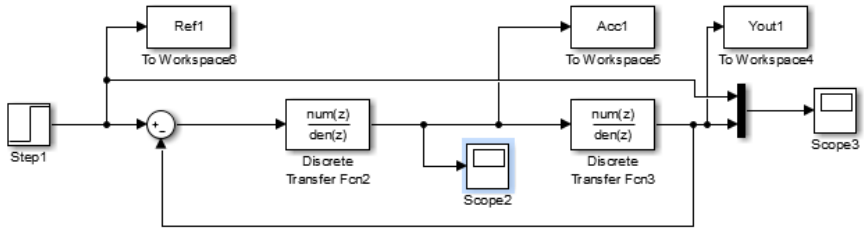


Figura 63. Esquema para la simulación del control en tiempo finito

Obteniendo los siguientes resultados:

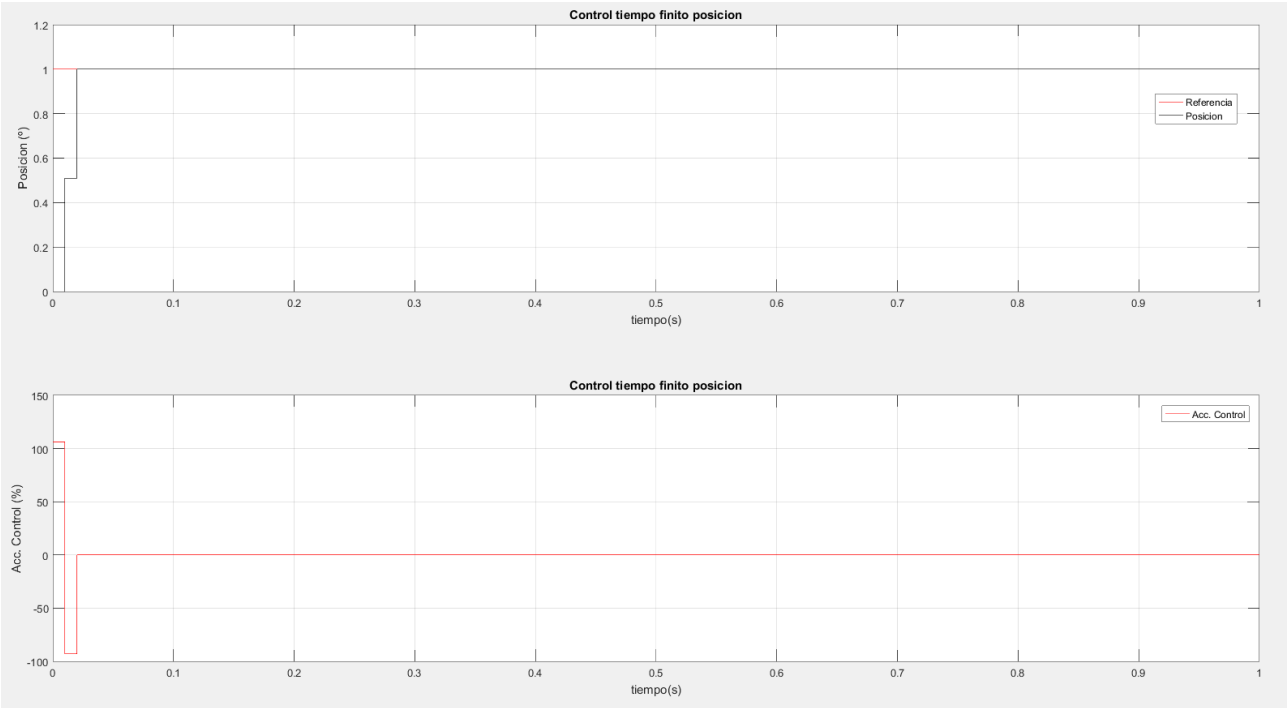


Figura 64. Respuesta del control en tiempo finito ante entrada en escalón en bucle cerrado

Donde, cómo se puede apreciar, se alcanza la referencia en apenas dos instantes pero con unas acciones de control demasiado grandes. Si analizamos las acciones de control:

$$U(z^{-1}) = \frac{M(z^{-1})R(z^{-1})}{G_p(z^{-1})} = \frac{(1-z^{-1})(1-0.8752z^{-1})}{0.0048z^{-1}(1+0.96z^{-1})} z^{-1}(1+0.96z^{-1})0.5102\frac{1}{1-z^{-1}}$$

$$U(z^{-1}) = \frac{M(z^{-1})R(z^{-1})}{G_p(z^{-1})} = \frac{(1-0.8752z^{-1})}{0.0048}0.5102$$

Por lo que las acciones de control serán finitas y no tendremos oscilaciones ocultas. Sin embargo, ante una referencia de 1º las acciones de control serían:

$$U(z^{-1}) = \frac{M(z^{-1})R(z^{-1})}{G_p(z^{-1})} = \frac{(1-0.8752z^{-1})}{0.0048}0.5102 = 106.29 - 93.026z^{-1}$$

Que en forma de secuencia:

$$\{U_k\} = \{106.29, -93.026\}$$

Excediendo la primera los límites de nuestro actuador. Acotaremos por ello esta primera acción de control, para que sea el valor máximo de control que podemos aplicar: 100. Al limitar esta primera acción de control aumentamos el orden de $M(z^{-1})$ en una unidad, por lo que aumentará en uno el número de instantes de control.

$$orden\big(M(z^{-1})\big) = 2 + 1$$

$$M_2(z) = d_0 + d_1 z^{-1}$$

$$M_1(z) = 1 + c_1 z^{-1} + c_2 z^{-2}$$

Donde:

$$1 - (z^{-1}(1+0.96z^{-1})(d_0 + d_1 z^{-1})) = (1-z^{-1})(1 + c_1 z^{-1} + c_2 z^{-2})$$

$$1 - d_0 z^{-1} - d_1 z^{-2} - 0.96 d_0 z^{-2} - 0.96 d_1 z^{-3} = 1 + c_1 z^{-1} + c_2 z^{-2} - z^{-1} - c_1 z^{-2} - c_2 z^{-3}$$

$$-d_0 = c_1 - 1$$

$$-d_1 - 0.96 d_0 = c_2 - c_1$$

$$-0.96 d_1 = -c_2$$

Obteniendo la cuarta y última ecuación de la acción de control:

$$U(z^{-1}) = \frac{M(z^{-1})R(z^{-1})}{G_p(z^{-1})} = \frac{\cancel{z^{-1}(1+0.96z^{-1})}(d_0 + d_1 z^{-1})}{\frac{0.0048\cancel{z^{-1}}\cancel{(1+0.96z^{-1})}}{\cancel{(1-z^{-1})}(1-0.8752z^{-1})}} \frac{1}{\cancel{1-z^{-1}}}$$

$$U(z^{-1}) = \frac{(1-0.8752z^{-1})(d_0 + d_1 z^{-1})}{0.0048} = \frac{d_0 + d_1 z^{-1} - 0.8752 d_0 z^{-1} - 0.8752 d_1 z^{-2}}{0.0048}$$

$$U(z^{-1}) = 208.33 d_0 + 208.33 d_1 z^{-1} - 182.33 d_0 z^{-1} - 182.33 d_1 z^{-2}$$

De donde nuestra primera acción de control deberá valer 100:

$$208.33 d_0 = 100;\ d_0 = 0.48$$

Se despejan el resto de parámetros:

$$c_1 = 0.52$$

$$c_2 = 0.02899$$

$$d_1 = 0.0302$$

Siendo el nuevo controlador:

$$G_r(z^{-1}) = \frac{1}{\frac{0.0048\cancel{z^{-1}}\,\cancel{(1+0.96z^{-1})}}{\cancel{(1-z^{-1})}(1-0.8752z^{-1})}} \frac{\cancel{z^{-1}}\cancel{(1+0.96z^{-1})}(0.48+0.0302z^{-1})}{\cancel{(1-z^{-1})}(1+0.52z^{-1}-0.02899z^{-2})}$$

$$G_r(z^{-1}) = \frac{(0.48+0.0302z^{-1})(1-0.8752z^{-1})}{0.0048(1+0.52z^{-1}-0.02899z^{-2})}$$

$$G_r(z) = \frac{(0.48z+0.0302)(z-0.8752)}{0.0048(z^2+0.52z-0.02899)}$$

Y las acciones de control:

$$U(z^{-1}) = 99.99 + 6.29z^{-1} - 87.5184z^{-1} - 5.50z^{-2}$$

Que en forma de secuencia:

$$\{U_k\} = \{99.99, -81.22, -5.50\}$$

La implementación se realizaría con la siguiente ecuación en diferencias:

$$u(k) = -0.52u(k-1) + 0.039u(k-2) + 100e(k) - 93.79e(k-1) - 5.5e(k-2)$$

Ecuación 43. Ecuación en diferencias del control en tiempo finito diseñado en forma directa

Si tomáramos una entrada en escalón de valor 2, como en el caso más desfavorable que generará nuestro perfil, deberíamos de nuevo realizar el ajuste de la acción de control teniendo en cuenta este valor y la acción de control máxima que podemos suministrar el motor. El ajuste de la acción de control resultaría en un procedimiento iterativo hasta dar con las condiciones óptimas si se cambiara la referencia y por ello, siendo la naturaleza y el objetivo del trabajo los de conseguir unas bases de partida para el futuro desarrollo, se desestimará este control como solución a desarrollar en el presente proyecto, no obstante se realizará un ensayo comprobando la respuesta del mismo ante una rampa controlada que presente escalones inferiores a 1º.

En cualquier caso, el procedimiento para la obtención del control para el modelo de velocidad sería el mismo cambiando el modelo.

4.7.5 Efecto de la perturbación

En control existen dos problemas fundamentales, el primero, el conocido como problema del servo es el que hemos estado abordando e intentando solucionar hasta ahora: el seguimiento de una referencia con un error (nulo o mínimo). El segundo problema es el de regulación y consiste en minimizar el efecto de las perturbaciones.

El error de salida del que nos informa la realimentación puede tener dos naturalezas: un cambio en la referencia, o una perturbación en régimen permanente. Si estudiado el efecto de la perturbación en nuestra variable a controlar éste es importante, debemos reducir su efecto.

Para ello, es necesario un modelo de la perturbación. Existen tres maneras de reducir este efecto:

- Reducción en la fuente.
- Realimentación local: control en cascada.
- Prealimentación.

Y un cuarto procedimiento, si la perturbación no es medible, es la reducción por predicción.

La perturbación que más afecta a los motores es la carga. Ésta es lenta y normalmente periódica. Podríamos, por tanto, clasificarla como determinista.

Un posible controlador para reducir el efecto de la perturbación que supondría una carga en el eje, medible a partir de la corriente que circula por el motor, se ha indicado ya en el apartado de modelado matemático del motor, y consistiría en una prealimentación de esta perturbación medible.

Para los casos en los que no se tuvieran estas características de la perturbación deberá ser tratada como estocástica (solo previsible mediante herramientas estadísticas).

Existen distintos modelos para el manejo de las perturbaciones estocásticas para el diseño de controladores basados en el ruido blanco (media móvil, ARMAX, etc). A partir de estos modelos podría diseñarse un regulador de mínima varianza que eliminara el efecto de la perturbación y que, en conjunto con un integrador, pudiera seguir una referencia de escalón.

Podría para ello realizarse una identificación experimental, tomando muestras de la velocidad partiendo de un punto de funcionamiento distinto de 0 para evitar la zona muerta y las no linealidades que conlleva y utilizando la herramienta de Matlab Ident para conseguir el modelo. A partir de ahí, el diseño de un regulador de mínima varianza consiste en aplicar su definición.

Sin embargo, una vez más, no se realizará este estudio en este trabajo por escapar a los objetivos del mismo.

4.8 Generación de trayectorias y patrones de movimiento

En el estudio de condicionantes se ha indicado que existen diferentes tipos de movimiento para sistemas robotizados y de control numérico, entre los cuales, el más utilizado es el movimiento punto a punto y concretamente el correspondiente a patrones o perfiles de movimiento en forma de curva trapezoidal o de curva en S.

En el contexto del movimiento punto a punto, una curva en S consta de 7 fases distintas de movimiento. En la fase I la carga comienza a moverse desde el reposo a una aceleración linealmente creciente hasta un valor máximo. En la fase II, la carga acelera de forma constante manteniendo este valor máximo. En la fase III comienza una deceleración lineal en la que la aceleración llega a 0, instante en el que la velocidad alcanza su valor máximo. La velocidad se mantiene constante durante la fase IV, y después la carga decelera de forma simétrica a las tres primeras fases en las fases V, VI y VII.

Los perfiles trapezoidales, por el contrario, tienen solo 3 fases correspondientes a fase I, aceleración constante, fase II, velocidad constante y fase III, deceleración constante.

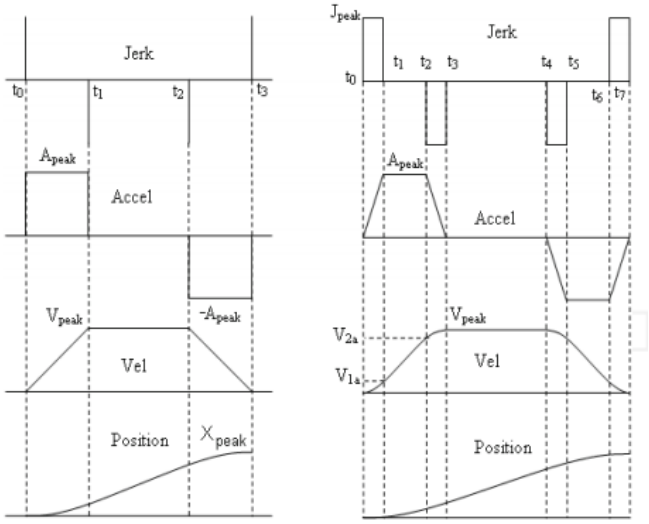


Figura 65: Curva en S trapezoidal (Izquierda) y curva en S polinomial de tercer orden (derecha).

Esta diferencia de fases marca la diferencia entre ambos perfiles, resultando las fases adicionales presentes en el perfil en S en unas transiciones entre periodos de aceleración y no aceleración más suaves. Por el contrario, las transiciones del perfil trapezoidal son instantáneas.

Las características de movimiento asociadas a las variaciones de la aceleración se denominan "*jerk*". Se define como la variación de la aceleración respecto al tiempo, si el intervalo es lo suficientemente pequeño se traduce en su derivada. En los perfiles trapezoidales, el *jerk* es infinito en las transiciones, lo que se traduce en una discontinuidad transmitida a la carga como un rebote. En los perfiles en S, el *jerk* es constante, y la energía asociada al cambio de la aceleración se libera a tiempo, eliminando ese enganche del caso anterior.

Así pues, variar la aceleración de forma gradual posee varias ventajas, una de las cuales es la menor oscilación de la carga.

Para una carga dada, a mayor *jerk*, mayor la vibración indeseada, pudiendo estropear completamente el control y la precisión del mismo si se alcanza el estado de resonancia.

Como los perfiles trapezoidales funcionan con aceleración y deceleración máximas son, desde el punto de vista de la ejecución, más rápidos que los perfiles en S. Sin embargo, si la aproximación todo o nada causa un incremento en el tiempo de establecimiento, esta ventaja podría perderse. Afortunadamente, esto puede arreglarse e incluso optimizarse el rendimiento añadiendo una pequeña transición entre periodos de reposo y aceleración.

En la mayoría de aplicaciones, solo una pequeña parte de las transiciones entre aceleración y no aceleración puede reducir substancialmente la vibración. Así, es posible optimizar el rendimiento ajustando la curva en S para la carga dada y la velocidad deseada.

En el análisis de la aplicación, la mejor forma de perfil en S dependerá de la naturaleza mecánica del sistema y de las especificaciones del funcionamiento deseado. Por ejemplo, en aplicaciones médicas que incluyen transferencia de líquidos que no deben ser derramados, sería apropiado elegir un perfil que carezca de la fase II y VI. El perfil ideal en este caso es el que alarga las transiciones de aceleración tan lejos como sea posible, consiguiendo con ello maximizar la suavidad del movimiento.

En aplicaciones de "*pick-and-place*" a alta velocidad, por el contrario, la velocidad es el aspecto más importante, una buena opción sería un perfil en el que las fases de transición (I, III, V y VII) sean entre el 5 y el 15% de las fases II y VI. En este caso, el perfil añadiría una pequeña

cantidad de tiempo al total del movimiento, pero la reducción de la oscilación al final del mismo reduciría considerablemente este tiempo respecto a otros casos.

En cualquier caso, la meta de cualquier perfil de movimiento es alcanzar las características del sistema para la aplicación deseada. Los perfiles de curvas trapezoidales y en S funcionan bien cuando la curva de respuesta del par del sistema es lo bastante plana; esto sucede cuando el par de salida no varía mucho respecto al rango de velocidad esperado. Esto es cierto para la mayoría de sistemas basados en servomotores de corriente continua, ya sean con o sin escobillas

Los motores paso a paso, sin embargo, no poseen curvas de par-velocidad planas. El par de salida no es lineal, en ocasiones tiene una gran caída en la zona de "inestabilidad de medio rango", y si no es ahí, es seguro que la tendrá a altas velocidades. Esta inestabilidad ocurre a frecuencias de paso en las que la resonancia natural del motor es alcanzada por la frecuencia de variación de los pasos. Una forma de evitarlo es utilizar una velocidad de partida distinta de 0 para el perfil. No obstante, normalmente se requerirá un perfil parabólico para los casos de caída de par a altas velocidades, ya que este perfil presenta su menor aceleración cuando la velocidad es la más alta. Sin embargo en estos perfiles no hay transición como en las curvas en ese y que suavicen el cambio, por lo que si la oscilación de la carga debe tenerse en cuenta, esta solución será pero que el perfil en S, a pesar del hecho de que estos perfiles no están optimizados para motores paso a paso.

Vamos, a continuación, a presentar los distintos perfiles de movimiento.

4.8.1 Perfil de movimiento trapezoidal

Como se ha indicado antes, una forma típica de controlar articulaciones robóticas y otras configuraciones de elementos móviles es utilizar un perfil de velocidad de curva en S trapezoidal, que consiste en tres etapas: una aceleración, una velocidad máxima y una deceleración. La posición se define a partir de polinomios de segundo orden por lo que a veces se le llama polinomial de segundo orden.

$$a = \begin{cases} A_{peak}, & t_0 \le t \le t_1 \\ -A_{peak}, & t_2 \le t \le t_3 \end{cases}$$

Las matemáticas detrás de los perfiles de curvas trapezoidales son sencillas. Existen dos formas que se pueden utilizar: la forma continua (forma de ecuaciones físicas) y la forma discreta, utilizada en la mayoría de sistemas de móviles que utilizan microprocesadores para generar los nuevos parámetros de movimiento en cada cuenta del reloj.

Forma continua:

$$p(t) = p_o + v_o t + \frac{1}{2} a t^2$$

$$v(t) = v_o + at$$

$$a(t) = a$$

$$p_{decel} = \frac{v^2}{2a}$$

Forma discreta:

$$p(k) = p_{k-1} + v_k$$

$$v(k) = v_{k-1} + a$$

Ecuación 44. Modelo de patrón de movimiento trapezoidal

Donde p_o y v_o son la posición y velocidad iniciales; p_k y v_k son la posición y velocidad en el instante k y a es la aceleración.

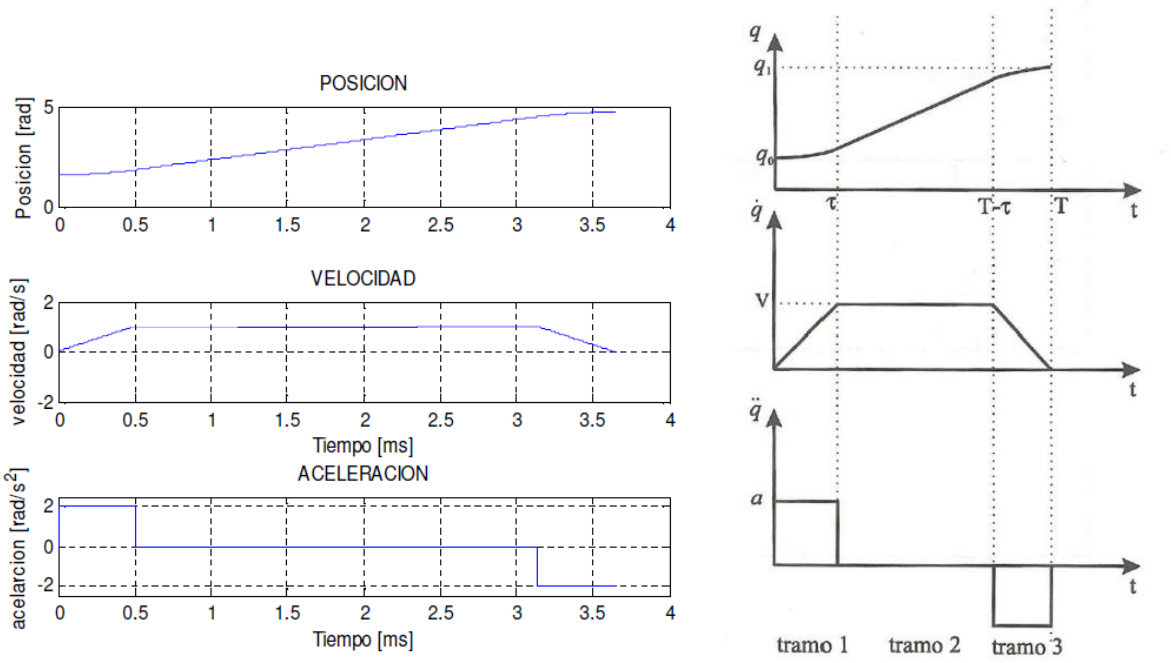


Figura 66. Ejemplos de perfil trapezoidal.

4.8.2 Perfil de movimiento de curva en S. Aproximación de tercer orden

Para los modelos polinomiales de orden mayor a dos los *jerks* presentan valores finitos, por tanto el perfil de la velocidad es más suave en el movimiento. Para el tercer orden el número de segmentos a conectar es 8. Si llamamos al perfil de velocidad de la trapezoidal M2, para el tercer orden tenemos:

$$j = M_3 = \begin{cases} M_2, & t_0 \le t \le t_3 \\ -M_2, & t_4 \le t \le t_7 \end{cases}$$

Forma continua:

$$p(t) = p_o + v_o t + \frac{1}{2} a_o t^2 + \frac{1}{6} j t^3$$

$$v(t) = v_o + a_o t + \frac{1}{2} j t^2$$

$$a(t) = a_o + jt$$

Forma discreta:

$$p(k) = p_{k-1} + v_k$$

$$v(k) = v_{k-1} + a_k$$

$$a(k) = a_{k-1} + j$$

Ecuación 45. Modelo de patrón de movimiento de curva en S de tercer orden

Donde p_o , v_o y a_o son la posición, velocidad y aceleración iniciales; p_k, v_k y a_k son la posición, velocidad y aceleración en el instante k y j es el jerk (variación en el tiempo de la aceleración).

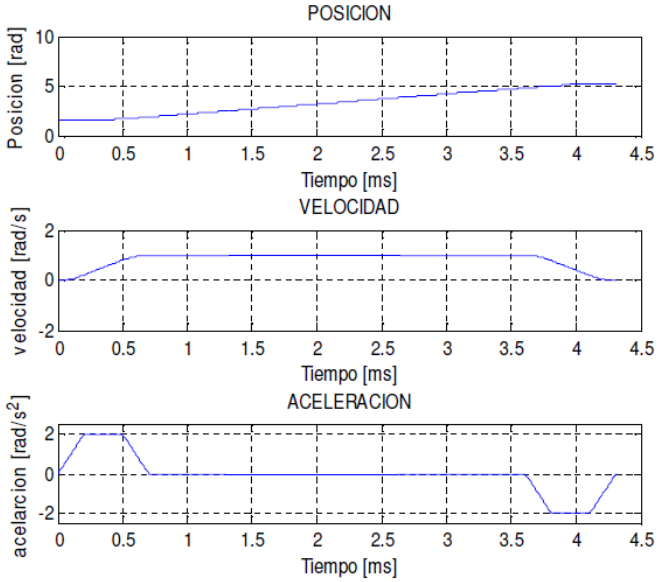


Figura 67. Perfil de curva de tercer orden.

Sin fases de aceleración máxima:

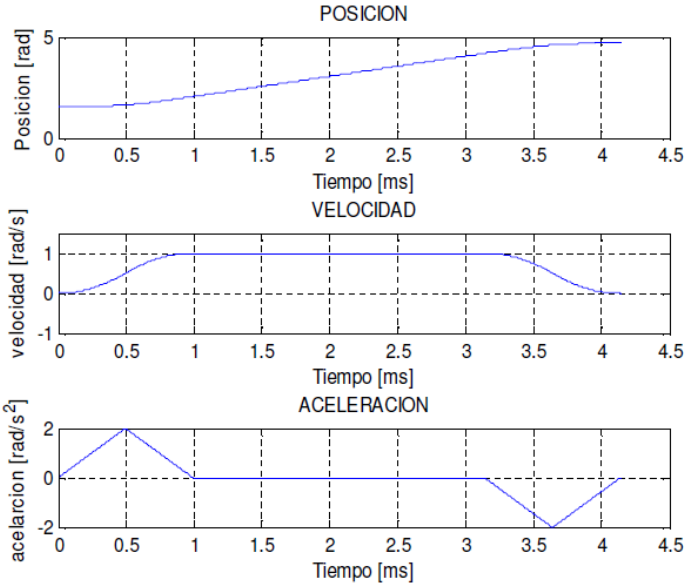


Figura 68. Perfil de curva de tercer orden sin fase de aceleración máxima.

4.8.3. Perfil de movimiento de curva en S. Aproximación de cuarto orden y orden n

De forma similar al planteamiento de las curvas de tercer orden, las de cuarto poseen 16 segmentos o fases que definir. Su planteamiento llamando M3 al perfil de la curva de tercer orden

es:

$$M_4 = \begin{cases} M_3, & t_0 \le t \le t_7 \\ -M_3, & t_8 \le t \le t_{15} \end{cases}$$

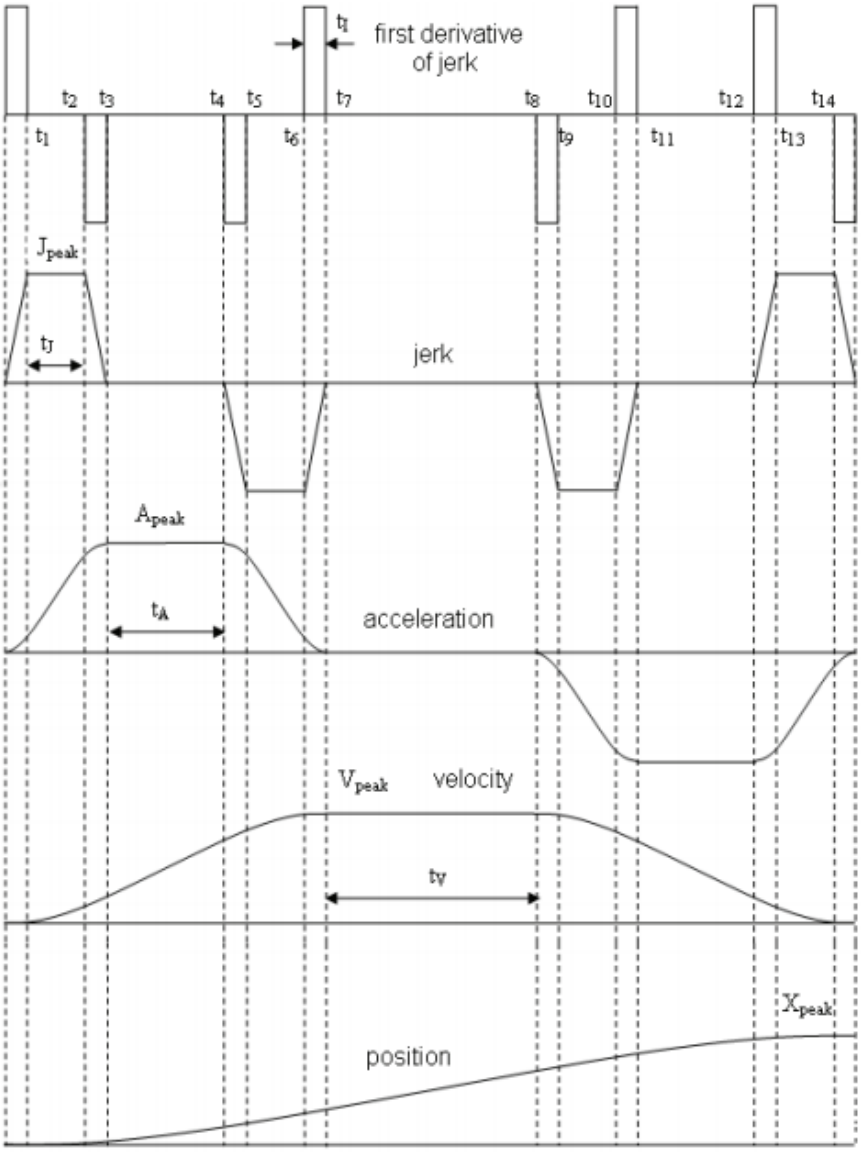


Figura 69. Perfil de curva de cuarto orden.

Y en general, para una curva polinomial de orden N tendremos 2^N fases, en las que la aceleración tendrá la forma por tramos de la velocidad del perfil anterior:

$$M_n = \begin{cases} M_{n-1}, & t_0 \le t \le t_{2^{n-1}-1} \\ -M_{n-1}, & t_{2^{n-1}} \le t \le t_{2^n-1} \end{cases}$$

4.8.4. Perfil de movimiento senoidal

Las curvas senoidales están relacionadas con las curvas en S, siendo también curvas de tercer orden, pero con la salvedad de que la aceleración responde a una función senoidal. Estas suponen un caso particular de curvas parabólicas, muy aplicadas para motores paso a paso.

Forma continua:

$$p(t) = p_o + v_o t + \frac{1}{2} a_o t^2 + \frac{1}{6} j t^3$$

$$v(t) = v_o + a_o t + \frac{1}{2} j t^2$$

$$a(t) = a_{max} sen(\omega t)$$

$$j(t) = j_{max} cos(\omega t)$$

Ecuación 46. Modelo de patrón de movimiento senoidal

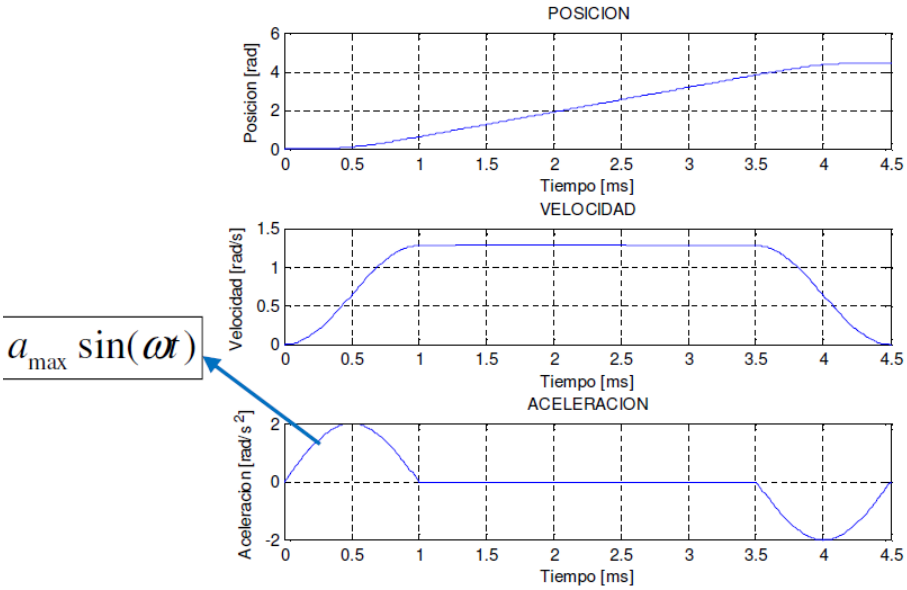


Figura 70. Perfil de curva senoidal

Por la naturaleza académica del proyecto y por simplificar los cálculos, la implementación y la labor del procesador, en términos de implementación nos serviremos del perfil trapezoidal como patrón para la generación de nuestras referencias.

5. Descripción detallada de la solución adoptada

5.1 Sistema de trabajo

El sistema que se utilizará para el ensayo experimental de los controladores y la validación de los mismos mediante la realización de movimientos consiste en un eje prismático.

Este posee una estructura formada por varillas roscadas de acero cincado unidas por cuatro esquinas impresas en PLA mediante impresión 3D. Sobre las esquinas descansan dos varillas lisas de acero rectificado que permiten el deslizamiento de dos rodamientos lineales (LM8uu) unidos a un carro que es la pieza móvil. En los extremos del eje se encuentran el motor**,** sujeto por una pieza impresa y, en el extremo opuesto, una polea formada por dos rodamientos biselados (T623zz) también sujeta por una pieza impresa que permite regular la tensión de la correa GT2 que supone la transmisión del movimiento del motor al carro.

El carro (pieza móvil) lo forma una lámina rectangular de metacrilato cortado por láser. Se incluye también otra lámina en uno de los laterales sobre la que se ha dispuesto papel milimetrado sirviendo la misma de referencia para la comprobación de resultados. La unión de elementos se ha realizado mediante tornillería, arandelas y tuercas de rosca métrica. La sujeción de piezas que requieren menor rigidez se realiza mediante bridas.

Todo el sistema es de diseño y elaboración propia, inspirado en el diseño del eje Y de la Prusa i3. Los planos se encuentran en el respectivo documento del presente proyecto y se adjuntan, en la versión electrónica, los ficheros .stl con las mallas para la impresión 3D de las piezas, así como los archivos .step elaborados en Solidworks para la posible y futura modificación. La siguiente figura muestra el diseño 3D de la estructura:

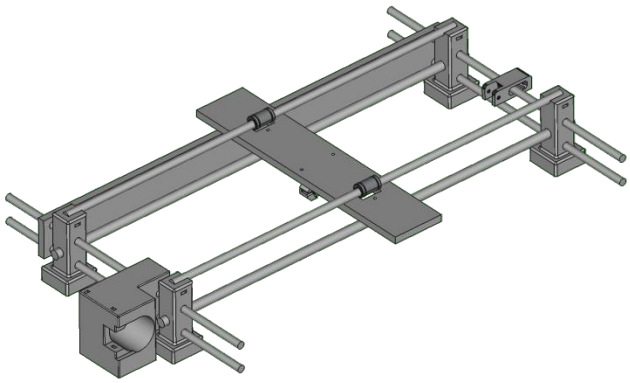


Figura 71. Modelo 3D del sistema de ensayos

El aspecto final del montaje es el siguiente:

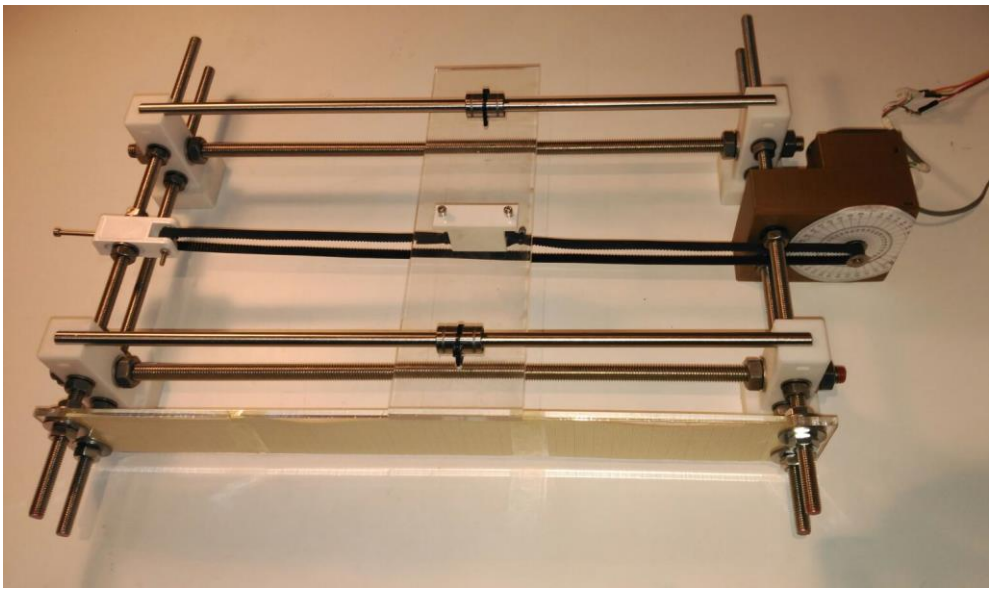


Figura 72. Implementación del sistema de ensayos

En cuanto al hardware electrónico, se dispone de un motor Pittman con encoder incremental de cuadratura E30 y una reductora en forma de engranajes planetarios G30A. Una etapa de potencia formada por una placa basada en el doble Puente-H L298. Un microcontrolador STM32f429I, y una fuente de alimentación para suministrar la tensión y corriente requeridas por el motor.

Además se dispone de un ordenador con sistema operativo Windows 7 para la programación del microcontrolador y de un osciloscopio digital para la comprobación de las señales.

Podemos obtener, para nuestro sistema, la relación entre posición lineal y angular al multiplicar la posición angular por el radio de la polea del motor y utilizando el factor entre los dientes de la correa GT2 y la polea según la referencia en [Reprap 16].

$$\frac{13\ dientes}{1\ vuelta}\frac{2mm}{diente} = \frac{26\ mm}{vuelta\ de\ eje}$$

Si la resolución mínima es de 1º:

$$1°\frac{26\ mm}{360°} = 0.072\ mm\ de\ precisión\ de\ movimiento\ lineal$$

Ecuación 47. Precisión de movimiento lineal del sistema

Lo que para un recorrido máximo de 270 mm del eje significa un total de movimiento angular de 3750 º.

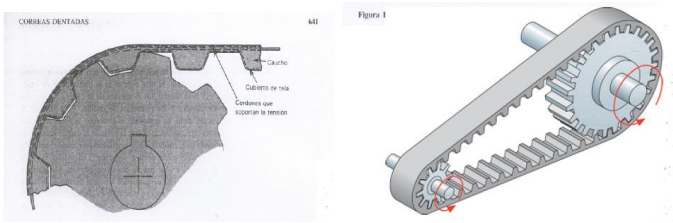


Figura 73. Detalle del sistema de transmisión por correa dentada.

5.2 Índices de rendimiento

Se van a simular e implementar una serie de controladores para poder comparar la respuesta del sistema ante cada uno. La forma de discernir el mejor funcionamiento serán los índices de rendimiento.

Un índice de rendimiento es un número que indica la bondad del comportamiento de un sistema. Así, se puede considerar que un sistema de control es óptimo si los valores de los parámetros se eligen de forma que el índice de comportamiento elegido sea máximo o mínimo, según el caso.

Un buen índice debe brindar selectividad, es decir, la capacidad de diferenciar un ajuste óptimo de los parámetros, claramente, de los ajustes no óptimos. Además, un índice debe presentar un único valor numérico positivo o cero (se obtiene un valor de cero si y solamente si la medida de la desviación es idénticamente cero).

En el presente proyecto se propone utilizar 2 índices para poder analizar el control desarrollado, aplicándose tanto al control de posición, como al de velocidad, y como al de fuerza.

Estos son:

- **Criterio Integral de Error Cuadrático:** De acuerdo con el criterio integral de error cuadrático (CIEC), la calidad de comportamiento del sistema se evalúa por medio de la siguiente expresión:

$$\int_0^{\infty} t^{\beta} e^2(t) dt$$

Donde:

$$\beta \geq 0$$

$$e(t) = ref(t) - y(t)$$

En este índice el límite superior ∞ se puede sustituir por *T*, que se elige lo suficientemente grande como para que *e(t)* sea despreciable para *T<t*. El control óptimo será el que minimice esta integral. Además, este índice da una mayor importancia a los errores grandes y menor a los pequeños cuando β=0, mientras que para β≥1 un error inicial grande tiene poco peso respecto de los errores que se producen más tarde en la respuesta transitoria

- Criterio Integral de Error Absoluto: Con el criterio del error absoluto (CIEA), tal como especifica su nombre, el índice estará en función del error absoluto entre la referencia deseada del proceso y la salida obtenida:

$$\int_0^{\infty} t^{\beta} |e(t)| dt$$

Donde:

$$\beta \geq 0$$

$$e(t) = ref(t) - y(t)$$

Al igual que antes, el límite superior ∞ se puede sustituir por *T*, que se elige lo suficientemente grande como para que *e(t)* sea despreciable para *T<t*. En este proyecto se propone implementar un control digital de un motor de corriente continua. Por ello el cálculo de dichos índices de comportamiento (asumiendo además que β=0) se puede aproximar mediante las expresiones siguientes:

$$CIEC = \sum_{i=0}^{n} e(k)^2$$

$$CIEC = \sum_{i=0}^{n} |e(k)|$$

Ecuación 48. Índices de rendimiento a implementar

5.3 Implementación

5.3.1 Disposición y conexiones del hardware

En nuestra solución dispondremos de dos medidas de salida del motor, una de intensidad y otra de posición.

La intensidad la proporciona el pin 1 de nuestro L298, si nos fijamos en la hoja de características en el anexo del proyecto [ANEXO A.2.4]. Como podemos apreciar, la función de este pin y del número 15, en el caso de controlar otro motor al mismo tiempo, es la de servir como sensor de la corriente que pasa por la carga, en este caso, nuestro motor. Conectando una resistencia entre el pin y masa podemos conocer la corriente que está consumiendo nuestro motor. Tal y como indica la hoja de características, la tensión de salida oscilará entre -1 y 2 V. Se recomienda por el fabricante el uso de una resistencia de 0.5 Ω para la conexión. Con esta resistencia, toda la corriente se desviará a tierra, quedando protegida la entrada a nuestra tarjeta de adquisición de datos, además al tratarse de una tensión máxima de 2 V es compatible y seguro con los pines TTL de la misma.

Mediante un conversor A/D (pin PA1) podremos conocer en cada momento la corriente que circula por la carga simplemente dividiendo por el valor de la resistencia (0.5 Ω). El muestreo, implementado mediante DMA, se describirá más adelante.

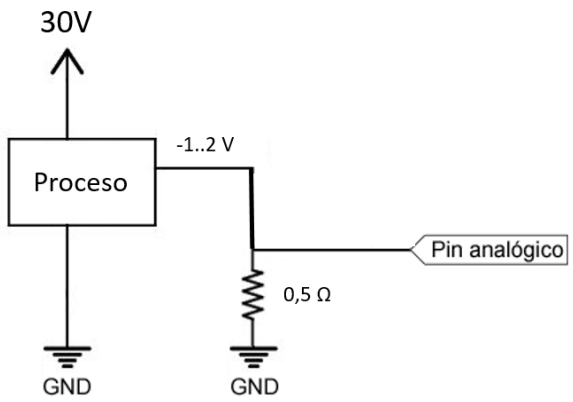


Figura 75. Esquema de conexión de la medición de corriente.

La medida de posición procede de un *encoder* incremental E30 Pittman compatible con nuestro motor y unido al eje del mismo por la parte trasera. Este *encoder*, como puede verse en el anexo [ANEXO A.2.3], posee una resolución de 360 pulsos por revolución, resolución que podemos doblar si en lugar de contar solo los pulsos de subida contamos también los de bajada, resultando en 720 pulsos por vuelta. Esto nos da una precisión de:

$$\frac{360\ º}{1\ rev}\frac{1\ rev}{720\ pulsos} = 0.5\frac{º}{pulso}$$

Ecuación 49. Precisión del *encoder*.

Esto es contando solo los pulsos de uno de los canales y teniendo como referencia el segundo para saber si el movimiento se realiza en un sentido o en el contrario, de manera que la variable contador que lleva asociada aumente o disminuya. Si contáramos ambos canales podríamos llegar a conseguir una resolución de 1440 pulsos, pero es una precisión que no es necesaria para el objeto del presente trabajo. El muestreo implementado mediante interrupción se desarrollará más

adelante en la configuración de la STM32F429-DISCOVERY. La conexión entre *encoder* y tarjeta de adquisición se realizará como sigue:

Cable (color)	Pin
Negro	5V
Verde	GND
Rojo	PE9 (CH1)
Blanco	PE11(CH2)

Figura 75: Tabla de conexión entre *encoder* y microcontrolador

Por último la alimentación del control a través de la etapa de potencia se realizará conectando sus terminales a las salidas indicadas como *OUT1* y *OUT2* de la placa del L298. Esta placa se alimentará a 10V en la entrada +12V con la fuente de alimentación (prestar atención al limitador de corriente). La salida PWM del microcontrolador en los pines *PC6* y *PC7* se conectarán a los pines de la placa *in1* e *in2* respectivamente, y se comprobará que los pines de *enable* junto al pin *in1* de la placa se encuentren cortocircuitados.

Un diagrama con las conexiones puede consultarse en el apartado de planos de este trabajo (plano 1).

5.3.1 Configuración de la tarjeta

La puesta a punto de la tarjeta es una tarea compleja debido a las nuevas bibliotecas del driver HAL (*Hardware Abstraction Layer*) que implementan los dispositivos de ARM-CortexM de ST, ya que existe poca documentación y desarrollo del mismo, mientras que existe una inmensa cantidad de librerías de la implementación *Legacy* ST_*TM*.

Necesitamos disponer de cinco elementos fundamentales para poder llevar a cabo nuestro proyecto en la tarjeta, estos son:

-Configuración del reloj de la tarjeta e inicialización.

-Configuración del *timer* que llevará la cuenta del muestreo.

-Configuración, inicialización y manejo del *encoder*, que será administrado por interrupción.

-Configuración, inicialización y manejo del ADC, que será administrado por DMA.

-Configuración, inicialización y manejo del generador PWM, que actuará en segundo plano.

-**Configuración del reloj de la tarjeta e inicialización:**

La necesidad principal del microcontrolador para un correcto funcionamiento es una señal de reloj que gobierne el funcionamiento del procesador y los distintos periféricos. Podemos consultar la configuración de la misma en el código del anexo [ANEXO A1.5] dentro de una función que hemos llamado *SystemClock_Config()*. Lo primero en hacer es habilitar la alimentación

del reloj con *__HAL_RCC_PWR_CLK_ENABLE()*, posteriormente, habilitamos un regulador de la alimentación para optimizar el consumo cuando no se trabaja a máxima frecuencia.

Después se habilita, configura e inicializa el oscilador.

Se selecciona la pulsación que gobernará el reloj del sistema, en nuestro caso PLL, se configuran los divisores del reloj del sistema y se inicializa el reloj.

Además se incluye una función para este módulo que hará de manejador de los errores informando de que han ocurrido y llevando el programa a un bucle infinito para detener su ejecución.

Cuando vayamos a crear nuestro programa principal deberemos, dentro del *main()*, llamar primero las funciones que inicializarán nuestro sistema.

El ejemplo de este procedimiento se encuentra en el mismo anexo, en el segundo punto del mismo.

Tras los *includes* y las declaraciones pertinentes comienza nuestro *main()*, primero se llama a la función *HAL_Init();*,ya implementada en las librerías HAL y que lleva el micro al estado de *reset*, inicializando la interfaz flash e inicializando el contador *Systick*, que utilizaremos para controlar el tiempo de ejecución del programa.

A continuación, llamaremos a la función que hemos creado para inicializar el reloj: *SystemClock_Config().* Por último, llamaremos a cada una de las funciones que hemos definido en las librerías que hemos creado, y que explicaremos a continuación, que inicializan los distintos periféricos y funciones de las que haremos uso, así como habilitan los pines que utilizaremos de la tarjeta:

```
encoder_Init();
Pwm_Init();
Adc_Init();
PRBS_Init();
```

-Configuración del *timer* que llevará la cuenta del muestreo:

Para poder controlar de forma precisa el tiempo de muestreo de nuestro control nos serviremos de la función *Delay* que implementa la librería HAL. El anexo [ANEXO A1.6] incluye las funciones y el código que procederemos a explicar a continuación.

En la inicialización de HAL se inicializa el *timer Systick*. Esta es la fuente de tiempo base del programa que produce interrupciones a intervalos regulares de tiempo. La función *HAL_GetTick();* devuelve el número de milisegundos que ha contado *Systick* desde el inicio del programa.

De esta forma podemos conocer el instante de tiempo en el que nos encontramos en nuestra ejecución y podemos obtener incrementos de tiempo restando el valor de dos llamadas consecutivas a esta función. Por otro lado, la función *HAL_Delay(__IO uint32_t Delay)*, produce una espera durante el número de milisegundos indicados en la variable *Delay* al llamarla.

Ya que el periodo de muestreo que vamos a implementar es de 10 ms, el uso de estas funciones es suficiente para lograr una precisa implementación del periodo de muestreo, sin embargo, podemos implementar funciones que permitan obtener variables aún más precisas del tiempo aprovechando los 180 MHz de frecuencia del reloj que hemos inicializado, pero su implementación y desarrollo se deja para futuras mejoras del proyecto actual.

Un ejemplo de aplicación en un muestreo como el que realizaremos a lo largo del proyecto se encuentra en el mismo anexo en el segundo apartado. En resumen se hacen dos llamadas a

HAL_GetTick(), una al comienzo del bucle de control y otra después de la actualización de variables. La espera se realiza para que la iteración tarde exactamente el tiempo deseado, durante (10-el tiempo entre lecturas) ms.

-Configuración, inicialización y manejo del *encoder*, que será administrado por interrupción:

La lectura del *encoder* la realizaremos mediante un *timer*, cuyo contador incrementará o reducirá su cuenta por cada pulso que se detecte del *encoder* en una dirección o en otra. El anexo [ANEXO A1.7] contiene la función de inicialización *encoder_Init()* y la función de lectura *uint32_t encoder_Read()*, que configurará e inicializará el *timer* que administra el *encoder* y devolverá la lectura de la situación actual del *encoder* respectivamente.

Para la inicialización, primero debemos configurar los pines que utilizaremos para la lectura, en nuestro caso los pines 9 y 11 del puerto E: PE9 y PE11. Elegimos estos pines de acuerdo a las hojas de características de la tarjeta y su manual (los cuales no anexamos por el tamaño, pero incluimos en la bibliografía: [STM 15], [STM 16]) por ser los que disponen de los canales 1 y 2 del TIM1. De los 16 *timers* disponibles en la placa, el TIM1 y el TIM8 son los únicos clasificados como avanzados pues permiten unas características y funciones asociadas especiales. Pueden, no obstante, utilizarse como *timers* normales. Usaremos TIM1 para la lectura del *encoder* y TIM8 para la generación de la señal PWM.

La GPIO, (*General Purpose Input Output*) puede configurarse en varios modos, entrada, salida o función alternativa, asociada a un *timer*, conversor o alguna otra función. Además cada pin tiene posibilidad de configurarse en modo *Push-Pull*, pudiendo o no activar una resistencia de *Pull-Up* o *Pull-Down*; o en modo *Open Drain*. El esquema de la célula programable asociada a cada pin es este:

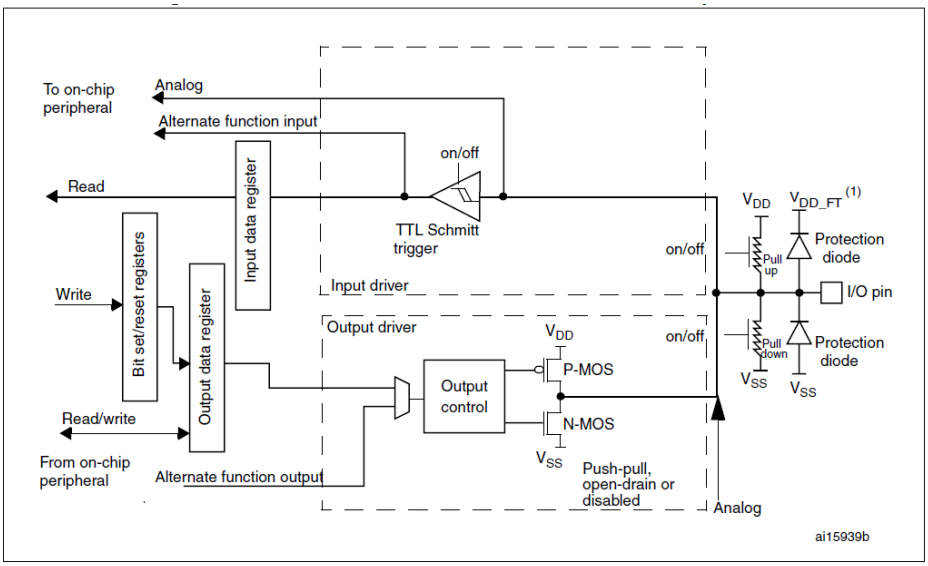


Figura 76: Célula general de configuración de los pines de entrada y salida para uso general de la placa

Tal y como se indica en el código, daremos reloj al periférico del puerto E (*__HAL_RCC_GPIOE_CLK_ENABLE()*)y al *timer* TIM1 (*__HAL_RCC_TIM1_CLK_ENABLE()*), inicializaremos la GPIO con los pines antes citados en modo función alternativa en configuración *PushPull*(*GPIO_MODE_AF_PP*), activando la resistencia de *Pull-Down* (*GPIO_PULLDOWN*), además indicaremos que la función alternativa del pin vendrá dada por el *timer* TIM1 (*GPIO_AF1_TIM1*). *HAL_GPIO_Init()* finaliza la inicialización de los pines.

A continuación, habilitamos el *timer* (*__TIM1_CLK_ENABLE()*), y configuramos sus

características. Configuramos la estructura que inicializará la lectura del *encoder* indicando que leeremos tanto flanco de subida como de bajada (*TIM_ICPOLARITY_BOTHEDGE*) e inicializaremos la estructura: *HAL_TIM_Encoder_Init()*. Por último haremos que la lectura de datos comience: *HAL_TIM_Encoder_Start()*.

Los tipos de datos que se utilizan para inicializar periféricos y configurar sus funciones son datos del tipo *struct* que HAL tiene predefinidos. Todos se definen de acuerdo a la biblioteca antes de ser utilizados en las funciones. Normalmente nada más empezar la función, o antes de la cabecera de la misma si tienen que ser globales para que otras funciones puedan acceder a ellos.

La función para leer datos del *encoder* que antes hemos presentado se encuentra después de la de inicialización y simplemente devuelve el valor de la cuenta al ser llamada.

En el segundo apartado del anexo puede verse un ejemplo de uso de esta biblioteca en el código para leer posición y velocidad del eje del motor. La posición se pasa a radianes multiplicando por 6.28/360, que es el valor de un pulso en radiantes, y la velocidad en dos instantes sucesivos, al ser el periodo de muestreo fijo, se obtiene dividiendo la diferencia entre dos valores consecutivos de la posición entre 0.01, resultando en rad/s. Tras el cálculo, el valor de la posición anterior se actualiza y termina la iteración.

Una mejora a implementar en el futuro será la administración por DMA de la adquisición de datos del *encoder*.

- Configuración, inicialización y manejo del ADC administrado por DMA:

Un conversor analógico digital es un elemento que nos permitirá pasar del valor de tensión en un pin de la tarjeta a uno digital entre 0 y $2^N - 1$, donde N es la resolución de nuestro convertidor.

El modo de funcionamiento más sencillo de implementar del ADC (*Analog to Digital Converter*) es mediante *polling*. El inconveniente que este plantea es que mientras se solicita la conversión, se efectúa y se devuelve el valor convertido, el procesador debe estar completamente dedicado a esta tarea. Esto supone una inversión de un muy preciado tiempo de muestreo que es limitado. Por ello, el siguiente nivel de implementación a considerar es mediante interrupción, de manera que cuando se detecte un cambio determinado en la entrada, se active una interrupción que administre la conversión, pudiendo hacer otras tareas sin un coste temporal tan alto. Sin embargo, si realmente se desea economizar en tiempo y optimizar la administración y adquisición de datos, la configuración adecuada, y la que se ha decidido implementar es un ADC gestionado por DMA. De esta forma un registro irá actualizando y guardando datos conforme se generen siendo la gestión y el proceso del ADC completamente ajena e independiente del procesador, con un coste computacional 0. Cuando se necesite un valor simplemente se tomará del registro de memoria en ese instante, mientras la conversión se efectúa continuamente de fondo.

En el anexo [ANEXO A1.4] encontraremos las funciones de la biblioteca creada para el uso del ADC.

En primer lugar tenemos declaradas las variables que almacenarán el valor final que mediremos, el número de medidas, el buffer donde guardaremos las muestras y el tamaño de este buffer *(ADCValue, MeasurementNumber, ADCBuffer y ADC_BUFFER_LENGTH* respectivamente).

A continuación se definen dos estructuras, la del ADC y la de la DMA, se definen como globales porque otras funciones tienen que hacer uso de ellas (*AdcHandle* y *DmaHandle*).

La función *Adc_Init()*, inicializa el ADC y la DMA junto con el resto de periféricos necesarios para su uso. En primer lugar, y de manera análoga al caso del *encoder*, se inicializa el pin 1 del puerto A (PA1) en modo analógico y sin resistencias de *pull up*/*down*. Después se habilita y configura el ADC, indicando la resolución de 12 bits en este caso (*ADC_RESOLUTION_12B*), en

modo continuo y como fuente de interrupción externa el *ADC_EXTERNALTRIGCONV_T1_CC1*, que esta codificado como 0. Se llama a la función *HAL_ADC_Init()* con la estructura configurada y se inicializa con ello el ADC.

Ahora se configura el canal, el pin que hemos elegido funciona con cualquiera de los 3 ADC (ADC123) y para el ADC1, lo hace en el canal 1 (*ADC_CHANNEL_1*). Elegimos el número de ciclos para tomar una muestra. Nosotros hemos elegido el menor posible: 3 ciclos. Siendo sinceros, trabajar por debajo de 28 ciclos es un nivel de precisión innecesario en nuestro proyecto, pero aprovechando que el gasto de procesamiento es nulo, podemos permitirnos este capricho e implementar el ADC en todo su potencial. Aplicamos la configuración con la función *HAL_ADC_ConfigChannel()* y pasamos a la configuración de la DMA.

Acudiendo a la bibliografía una vez más ([STM 15], [STM 16]), observamos que la DMA que hay disponible para el ADC es la DMA2, y concretamente para el ADC con nuestra configuración, el canal 0 del stream4 de la DMA2 (*DMA2_Stream4*, *DMA_CHANNEL_0*), indicamos que la dirección será del periférico a la memoria (*DMA_PERIPH_TO_MEMORY*), que trabajará en modo continuo circular (*DMA_CIRCULAR*), y con alta prioridad (*DMA_PRIORITY_HIGH*) e inicializamos llamando a la función *HAL_DMA_Init()*.

Unimos la función del ADC con la DMA mediante *__HAL_LINKDMA()*. Habilitamos la interrupción que maneje el muestreo *HAL_NVIC_EnableIRQ()* y comenzamos con el muestreo en DMA con la función *HAL_ADC_Start_DMA()*.

Tras estos pasos, nuestra función ha configurado, inicializado y puesto en marcha el ADC muestreando con DMA. A continuación podemos ver tres funciones que se encargan del tratamiento de la inmensa cantidad de datos que toma el convertidor debido a la precisión que hemos definido en la configuración. Las funciones *ADC_IRQHandler()* y *DMA2_Stream4_IRQHandler()* simplemente llaman al verdadero manejador detrás del control del ADC y la DMA, estos a su vez durante su funcionamiento van llamando a dos funciones conforme se llena el tamaño del buffer que hemos especificado en la función de *start*.

Concretamente ejecuta una llamada cuando ha completado la mitad del buffer a *HAL_ADC_ConvHalfCpltCallback()* y, cuando ha acabado de llenarse, a *HAL_ADC_ConvCpltCallback()*. Estas funciones nos permiten implementar un tratamiento de datos que aun mejore todavía más la medida y precisión de nuestra señal. Nosotros hemos implementado una función que calcule el valor medio de la mitad de los datos del buffer y actualice la variable que guarda el valor del registro que nosotros recibiremos al llamar a *ADC_Read(void)* para disponer del dato. De esta manera siempre tendremos un filtro software del posible ruido que tenga la señal o las variaciones que pueda introducir ocasionalmente el convertidor.

En el segundo apartado del anexo tenemos de nuevo un ejemplo de uso de la función para medir la corriente que circula por el motor.

- Configuración, inicialización y manejo del generador PWM, que actuará en segundo plano:

La forma en la que los *timers* generan salidas en la STM32F429I-Discovery es mediante modulación de ancho de pulso (*Pulse – Width Modulation*, PWM, en inglés).

Los *timers* que los STM32 implementan como hardware son bloques separados que pueden contar desde 0 hasta un valor dado activando varios eventos mientras tanto. En el modo PWM el *timer* controla la salida de uno o varios canales de salida (máximo 4). Cuando se alcanza por el contador un 0, el máximo valor, o un valor de comparación predefinido por nosotros para algún canal el valor de la salida de dicho canal puede cambiarse. Existen varias formas de configuración para definir los eventos que cambiarán el valor y como lo harán.

Por ejemplo, si queremos comparar una señal de entrada triangular y modular dos canales con ella para cambiar cuando se alcance un determinado valor T2 el canal 2 y cuando se alcance el valor 400 el canal 1, siendo el valor 500 coincidente con el final del periodo y comenzando el periodo a nivel bajo tendríamos:

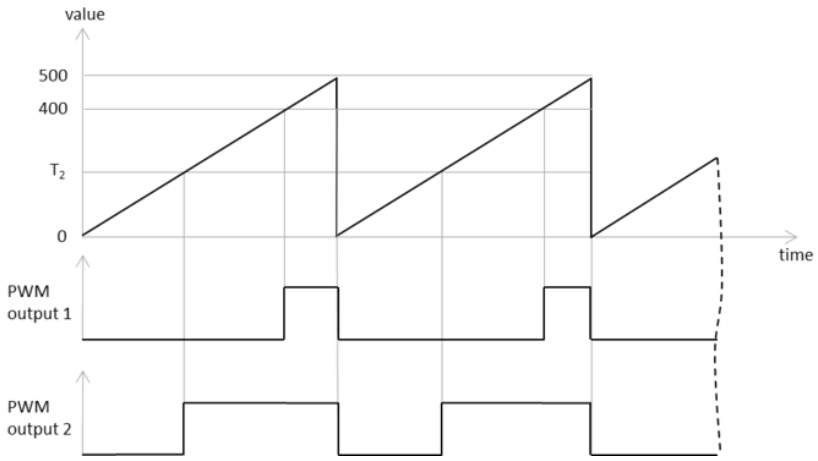


Figura 77: Ejemplo de la forma de generación de señal PWM

La ventaja de este tipo de modulación es que la salida se realiza sin ningún tipo de trabajo extra por parte de la CPU que puede dedicarse a otras tareas dentro de nuestro programa, sucediendo el PWM en *background*. Cabe destacar que podemos configurar y modificar el ciclo de trabajo para distintos canales pero que el periodo deberá ser el mismo por *timer*.

Antes de poder usar nuestro modo PWM necesitaremos configurar el *timer* para que empiece a contar. Lo configuraremos para que cuente hacia arriba con un periodo de 65535. Esto se muestra en el anexo [ANEXO A1.5] junto a un ejemplo de aplicación.

Declararemos como global la estructura de configuración del PWM para poder utilizarla con otras funciones. La función *Pwm_Init()* incluye todo lo necesario para configurar, habilitar y empezar a generar nuestra señal PWM. En primer lugar esta señal configura, como en los casos anteriores, la GPIO. Utilizaremos el TIM8 con sus canales 1 y 2. De acuerdo a [STM 15], [STM 16], los pines que incorporan la función alternativa para habilitar el periférico en ellos son el pin 6 y 7 el puerto C (PC6 y PC7) respectivamente. Damos reloj al puerto (*__GPIOC_CLK_ENABLE()*) y los configuramos en modo función alternativa *push pull* (*GPIO_MODE_AF_PP*), habilitando la resistencia de *pull-up* (*GPIO_PULLUP*), en alta velocidad (*GPIO_SPEED_HIGH*) e indicando que la función alternativa será la del *timer* 8 (*GPIO_AF3_TIM8*).

HAL_GPIO_Init() inicializa el periférico. Lo siguiente a configurar es el *timer*. Habilitamos la señal de reloj para el periférico como con el puerto anterior (*__TIM8_CLK_ENABLE()*), indicamos el *timer* a usar (*TIM8*), definimos el periodo de acuerdo con la explicación de los comentarios que se encuentran antes de la cabecera de la función de inicialización (*11250*), e indicamos que la cuenta será hacia arriba (*TIM_COUNTERMODE_UP*). Una vez configurado el *timer*, lo inicializamos en modo PWM con *HAL_TIM_PWM_Init()*.

Pasamos a configurar los parámetros de la señal en sí. Definimos un valor inicial en el que queremos que cambie la señal (en este caso empezaremos en el más bajo para que la señal sea 0 hasta que digamos lo contrario), indicaremos la polaridad o el nivel con el que empieza la señal el periodo, en nuestro caso un nivel alto (*TIM_OCPOLARITY_HIGH*) y el valor al que tiene que cambiar cuando se alcance el valor de comparación, en nuestro caso a nivel bajo o *reset* (*TIM_OCNIDLESTATE_RESET*). Haciendo uso de la función *HAL_TIM_PWM_ConfigChannel()* con la estructura que acabamos de configurar y en cada canal asignamos la configuración a los

mismos.

Por último configuramos los parámetros de tiempo muerto, bloqueo y parada que no debemos requerir ya que el control regulará con el ciclo de trabajo la señal conforme sea necesario, por ello deshabilitamos todos estos parámetros de configuración (*TIM_BREAK_DISABLE*, *TIM_LOCKLEVEL_OFF*, *TIM_OSSI_DISABLE*, *TIM_OSSR_DISABLE*). Actualizamos la configuración del *timer* con los nuevos parámetros mediante *HAL_TIMEx_ConfigBreakDeadTime()*. Y activamos la señal en ambos canales con *HAL_TIM_PWM_Start()*.

Se han implementado dos funciones más que hacen uso del manejador para modificar los parámetros de la señal cuando el control lo necesite. *pwm_Set()* actualizará el ciclo de trabajo al indicado y *pwm_Stop()* interrumpirá y deshabilitará la salida de la señal en los pines, se utiliza para parar el motor al final del programa.

Por último, en el segundo apartado del anexo se ejemplifica el uso de esta librería generando dos señales PWM y variando su ciclo de trabajo en dos bucles distintos.

La salida de la señal medida en el osciloscopio puede comprobarse en la siguiente figura:

Figura 78. Señale generadas por el microcontrolador

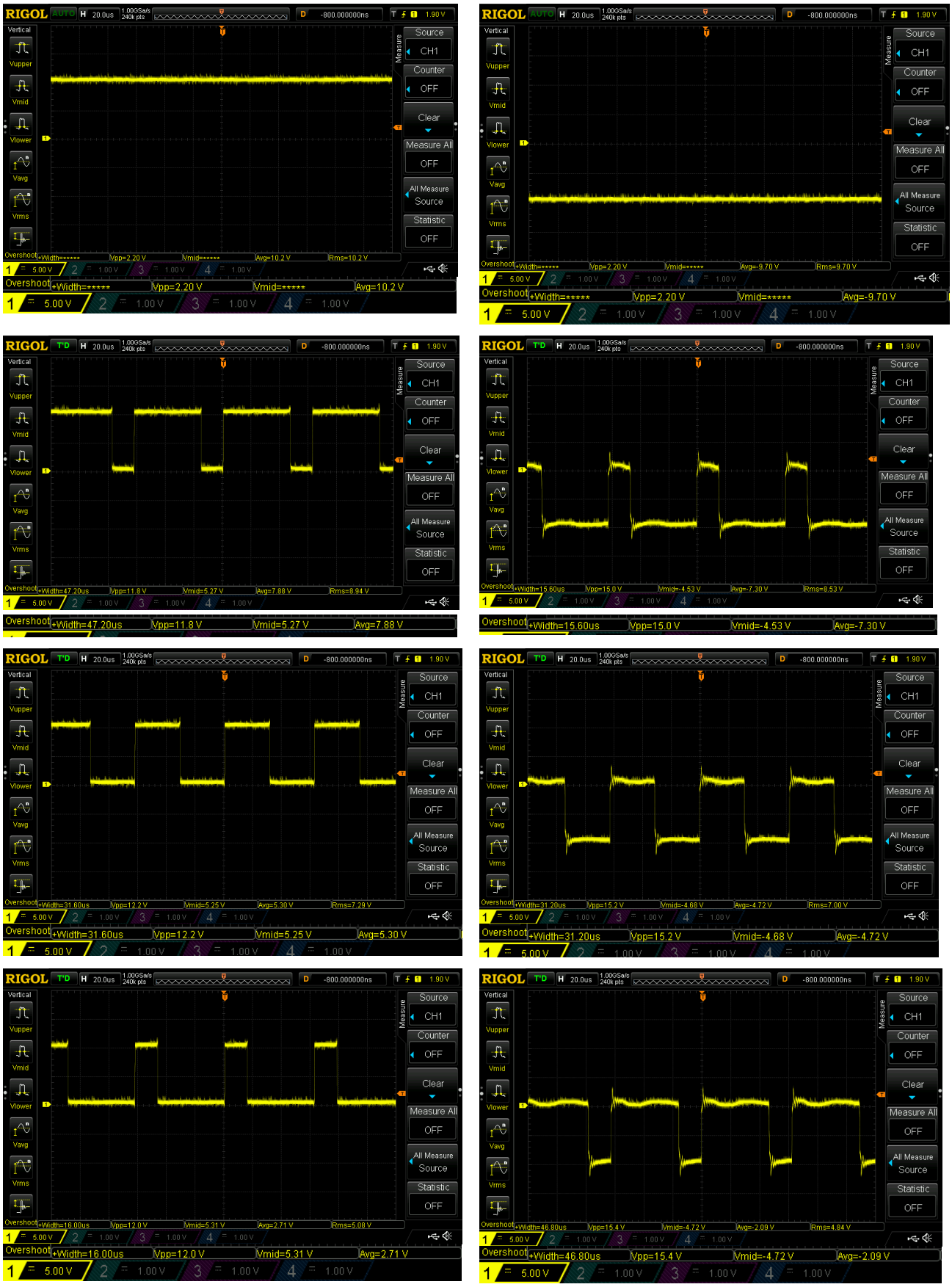


Figura 79. Señales a la salida de la etapa de potencia

5.3.2 Implementación del patrón de movimiento

En líneas generales, lo que buscamos es una función que nos devuelva referencias de velocidad, posición y aceleración al pasarle como parámetros la posición en la que nos encontramos, la posición que queremos alcanzar, la velocidad inicial en el intervalo, la velocidad máxima y la aceleración. Supondremos, no obstante, que la velocidad inicial en el intervalo es siempre 0, es decir, no concatenaremos movimientos pues realizamos movimientos punto a punto. En el anexo [A1.11] se incluyen los códigos de las bibliotecas implementadas para la generación de las referencias de nuestro control para movimientos basados en velocidad constante y velocidad a partir de un perfil de movimiento trapezoidal, de rampa, y de *spline* cúbica. También se incluye un script de Matlab para generar un perfil de curva en S que permita simular el comportamiento ante esta referencia. A continuación se muestran los resultados obtenidos al solicitar a estas funciones un movimiento entre 0 y 3750º siendo 800º/s la velocidad máxima de nuestro sistema y 1600º/s su aceleración:

Para movimiento de velocidad constante de 0 a 3750º:

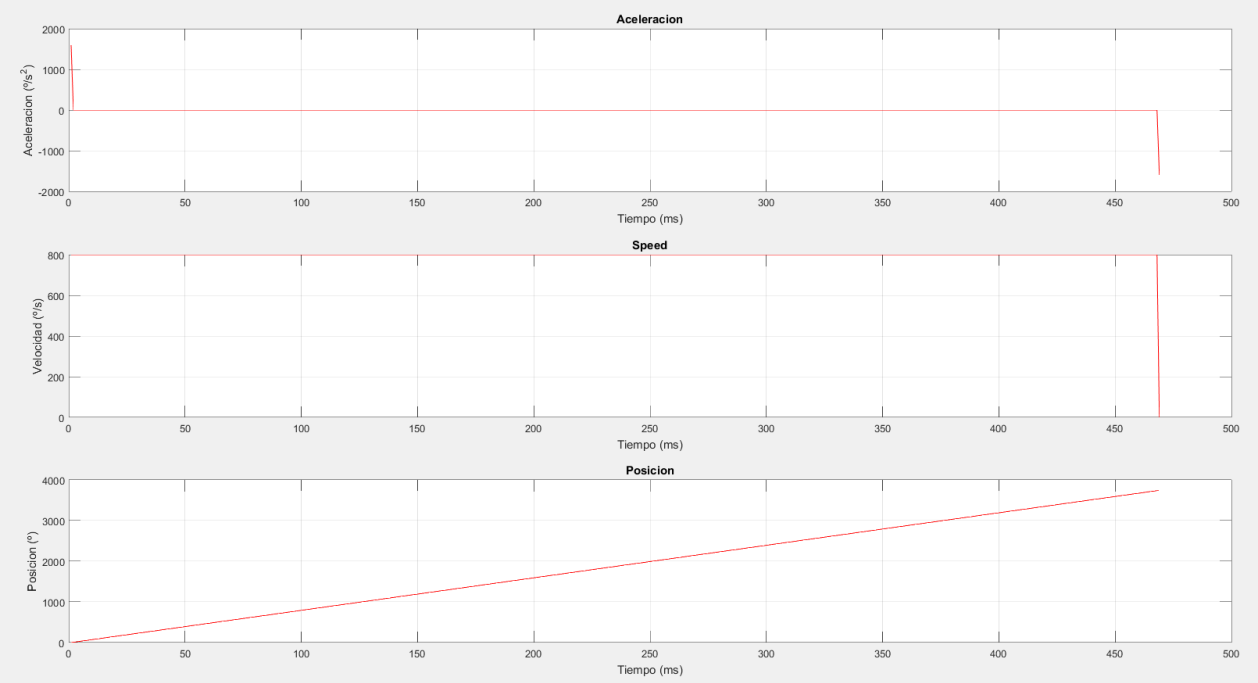


Figura 80. Perfil de movimiento en rampa para incremento positivo de posición

Para movimiento de velocidad constante de 3750 a 0º:

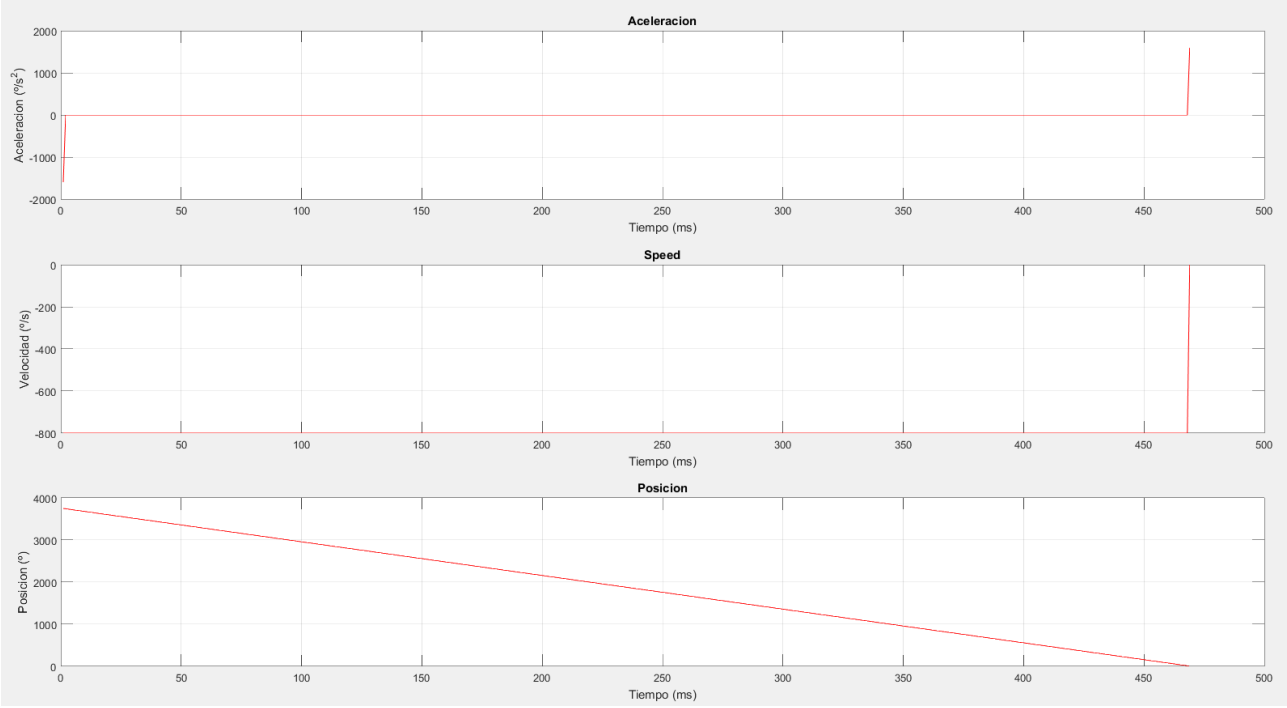


Figura 81. Perfil de movimiento en rampa para incremento negativo de posición

Para movimiento de velocidad según perfil trapezoidal de 0 a 3750º:

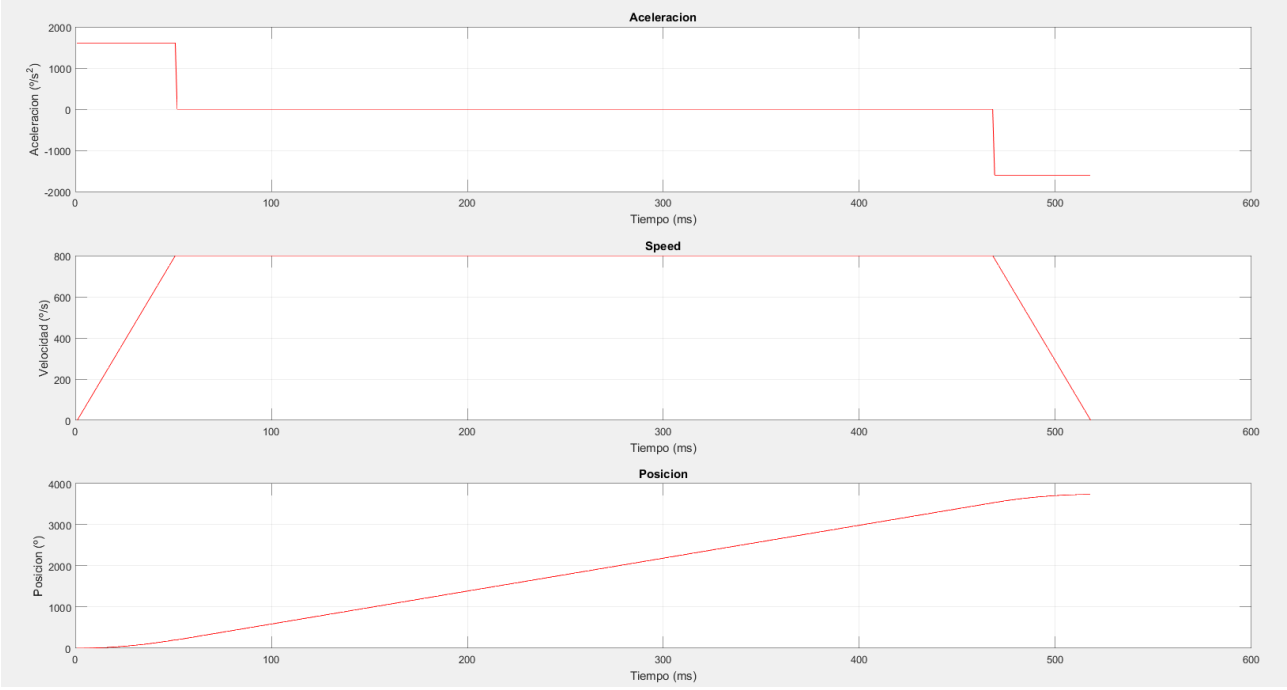


Figura 82. Perfil de movimiento trapezoidal para incremento positivo de posición

Para movimiento de velocidad según perfil trapezoidal de 3750 a 0º:

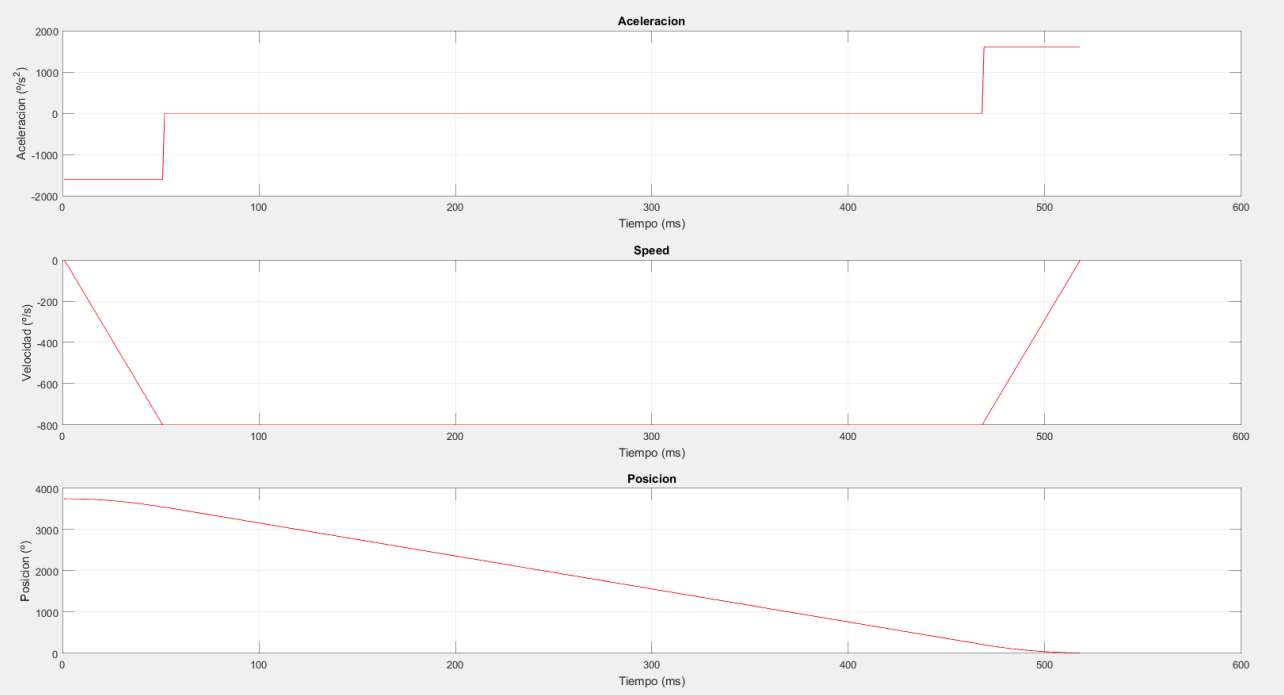


Figura 83. Perfil de movimiento trapezoidal para incremento negativo de posición

5.3.3. Diseño de los controladores a implementar. PID

Una vez analizados los distintos parámetros que se necesitan para cada controlador, para poder implementarlos en un algoritmo de control sólo hace falta saber el valor que se le tienen que asignar. Para ello existen varias opciones: estudio del lugar de las raíces, etc. En este caso se obtendrá a partir del método empírico de Ziegler-Nichols. De esta forma, los parámetros de los reguladores se podrán obtener a partir de los valores de *K*, *Tp* y *T0* obtenidos con el método de la respuesta a un escalón, o a partir de los valores de *Kc* y *Tc* calculados por el método de la respuesta sostenida [RAU 15].

Una ventaja importante de este método empírico es que no hace falta conocer el modelo del proceso. El inconveniente es que la sintonización no siempre logra un control adecuado y es preciso realizar un ajuste manual de los parámetros obtenidos para mejorar la respuesta.

En este método empírico hay dos maneras de obtener los parámetros: mediante la respuesta a escalón del proceso en bucle abierto, y mediante la respuesta oscilatoria mantenida en bucle cerrado. A continuación se detallan estos dos métodos.

- Respuesta a un escalón en bucle abierto: Es básicamente el mismo método utilizado en la identificación del proceso: se introduce un escalón a la entrada (unitario o no) y se mide el tiempo en que la salida del sistema tarda en alcanzar el 28,3% (*T1*) y el 63,2% (*T2*). A partir de estos tiempos se obtienen los parámetros *TP* y *T0* con las expresiones siguientes:

$$T_p = 1{,}5(t_2 - t_1)$$

$$T_0 = T_2 - T_p$$

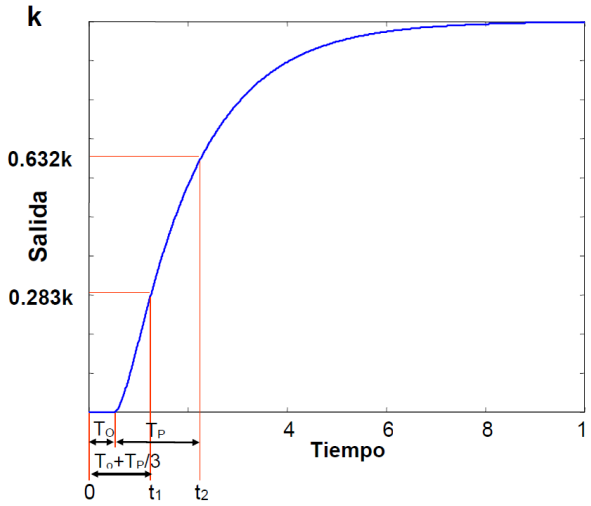


Figura 84. Criterio para la obtención de parámetros de PID mediante respuesta ante escalón

El último parámetro necesario para el diseño por sintonización es el valor de la ganancia K. Al igual que se ha comentado en el método de identificación, la ganancia del sistema será el cociente entre el valor final estable de la salida y el valor del escalón introducido.

Este método es muy interesante porque sólo hay que introducir un único escalón (a diferencia de lo que se verá posteriormente). La única precaución que se debe tomar en el caso que tengamos ruido en el sistema es que el valor del escalón de la entrada debe tener una magnitud elevada para minimizar la influencia de dicho ruido.

- **Respuesta sostenida:** En el método de la respuesta sostenida (o método de Ziegler Nichols) debemos cerrar el bucle de control con un regulador proporcional, de manera que se debe incrementar el valor de la ganancia proporcional hasta que la salida del sistema oscile con una amplitud constante. Con esta respuesta oscilatoria debemos obtener:

- *Kc*: valor de la ganancia para la cual el sistema oscila

- *Tc*: periodo (en segundos) de la respuesta oscilatoria del sistema

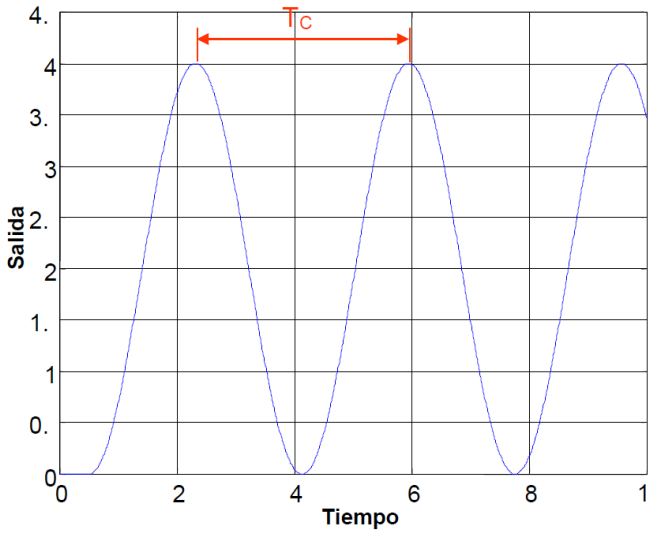


Figura 85: Identificación parámetros respuesta sostenida

Con este método se suelen obtener unos resultados más precisos que con el método de la respuesta a un escalón, aunque al ser un método de prueba y error se tarda más tiempo (como se ha comentado, se debe buscar el valor de *Kc* hasta que el sistema oscile). Además, tenemos el peligro de llegar a desestabilizar el sistema, o de no llegar a conseguir este comportamiento oscilatorio.

Los controladores se pueden obtener directamente a partir de la información obtenida con los parámetros significativos del método de la respuesta en bucle abierto y la oscilación mantenida anteriores, mediante la siguiente tabla:

Controlador	Parámetro	Respuesta Escalón	Respuesta Sostenida
P	K_P	$\frac{T_P}{KT_0}$	$0.5K_c$
PD	K_{PD}	$1.2\frac{T_P}{KT_0}$	$0.6K_c$
	T_d	$0.5T_0$	$T_c/8$
PI	K_{PI}	$0.9\frac{T_P}{KT_0}$	$0.45K_c$
	T_i	$T_0/0.3$	$T_c/1.2$
PID	K_{PID}	$1.2\frac{T_P}{KT_0}$	$0.6K_c$
	T_i	$2T_0$	$T_c/1.2$
	T_d	$0.5T_0$	$T_c/8$

Figura 86: Tabla de parámetros para la identificación del PID

Los valores de los parámetros obtenidos con la tabla son orientativos, por lo que siempre es conveniente realizar un reajuste manual para tratar de mejorar la respuesta. Una vez calculados los parámetros de los controladores continuos (*KP, Ti, Td*…) se pueden obtener los parámetros de los reguladores discretos (*q0, q1* y/o *q2*) mediante las expresiones mostradas en las ecuaciones anteriores.

Partiendo de los modelos de control obtenidos en el apartado de modelado y mediante los *scripts* de Matlab incluidos en el anexo [ANEXO A1.3] se van a obtener las constantes para implementar los controladores de tipo PID más adecuados para nuestros modelos. Una vez diseñados se implementarán en nuestro motor para comprobar y ajustar los resultados.

Para el modelo de velocidad, con el ajuste de respuesta ante escalón se tiene:

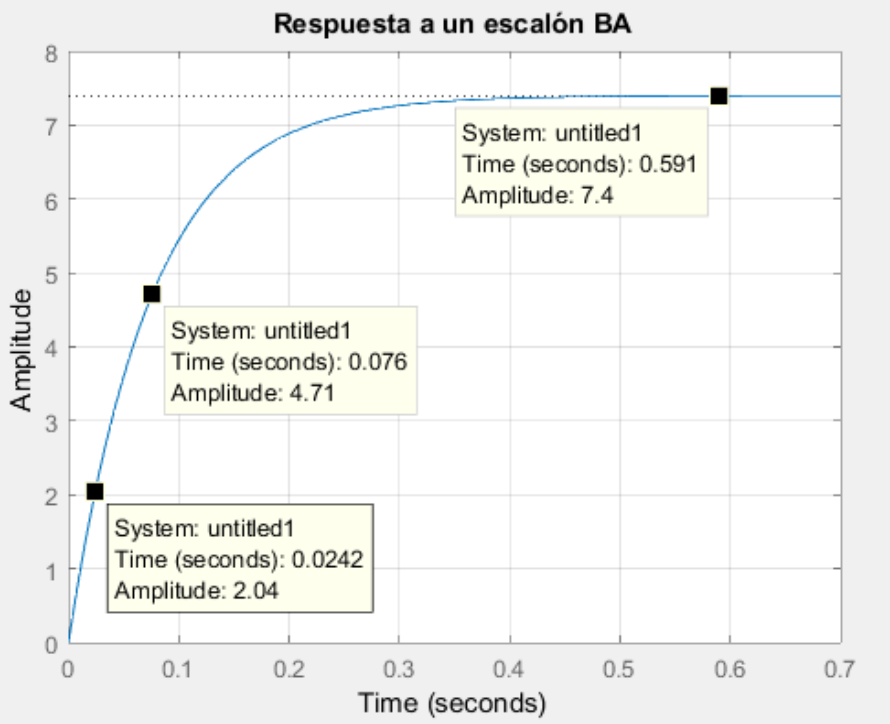


Figura 87: Identificación de parámetros de la respuesta ante escalón

Donde:

$$K = 7.4$$

$$k(T2) = 0.632 * 7.4 = 4.68 \rightarrow T2 = 0.076 + 0.01 = 0.086\ s$$

$$k(T1) = 0.283 * 7.4 = 2.09 \rightarrow T1 = 0.024 + 0.01 = 0.034\ s$$

Tenemos así que:

$$T_p = 0.078$$

$$T_0 = 0.008$$

Podemos ahora obtener los parámetros del control P, PD, PI y PID y comprobar su respuesta mediante simulación a partir del siguiente esquema de *Simulink*:

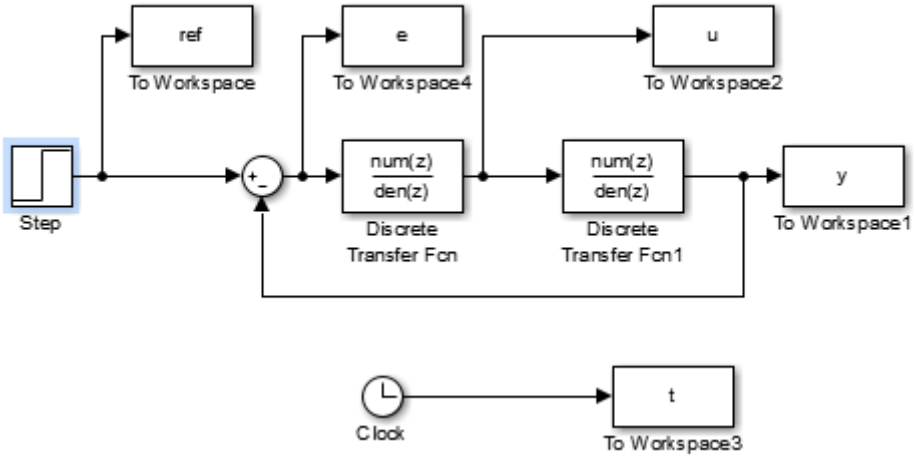


Figura 88: Esquema para la simulación del control diseñado

No debemos perder de vista en el transcurso del diseño de este y el resto de controles nuestros parámetros de respuesta deseados (tiempo de establecimiento inferior a 0.1s y sobreoscilación inferior al 5%). Se puede pensar en sacrificar alguno de los mismos a costa de alcanzar referencias más restrictivas, como en el caso de la velocidad escalones de mayor amplitud pero, teniendo en cuenta que el objetivo del proyecto es conseguir el movimiento deseado mediante la aplicación de referencias para el movimiento moderadas a partir de un perfil de respuesta suave no es interesante este sacrificio para la sobreoscilación, pudiendo ampliarse el tiempo de establecimiento para asegurar el cumplimiento de esta.

Si partimos de que la velocidad a máxima alimentación dentro de nuestro rango de trabajo, que se describe más adelante en el desarrollo de la solución adoptada, del motor con reductora es de aproximadamente 170 rpm, podemos considerar una referencia apropiada para el diseño del control mediante este procedimiento empírico.

$$\frac{170\,r}{min}\frac{360º}{1r}\frac{1min}{60s} = 1020\frac{º}{s}$$

Sin embargo, en el ensayo para obtener el modelo pudimos comprobar que la velocidad máxima real oscila en torno a un valor de 800º/s, por lo que tomaremos esta como referencia. Si se deseara alcanzar valores superiores sería necesario cambiar el punto de funcionamiento, por ejemplo, variando la tensión de alimentación a un valor superior (recordemos que este motor puede funcionar hasta una alimentación máxima de 48V).

Para los perfiles de movimiento que vamos a implementar como referencia el mayor incremento de velocidad que se puede solicitar como referencia en un instante es de 90º/s. Simularemos, por tanto, y ajustaremos los valores de nuestros controladores para este valor de referencia.

Pese a todos estos condicionantes la mayor limitación es la acción de control que puede solicitar el control y es que nuestro sistema está limitado a una acción de control entre el -100 y el 100 % del ciclo de trabajo. Con estas consideraciones y con los índices de rendimiento que se obtenga, se procede a estudiar los controladores para la velocidad.

Así, para el regulador proporcional, con la relación de parámetros de la tabla se tiene:

$$K_p = 1.3176$$

Con este valor, la respuesta del sistema en bucle cerrado es:

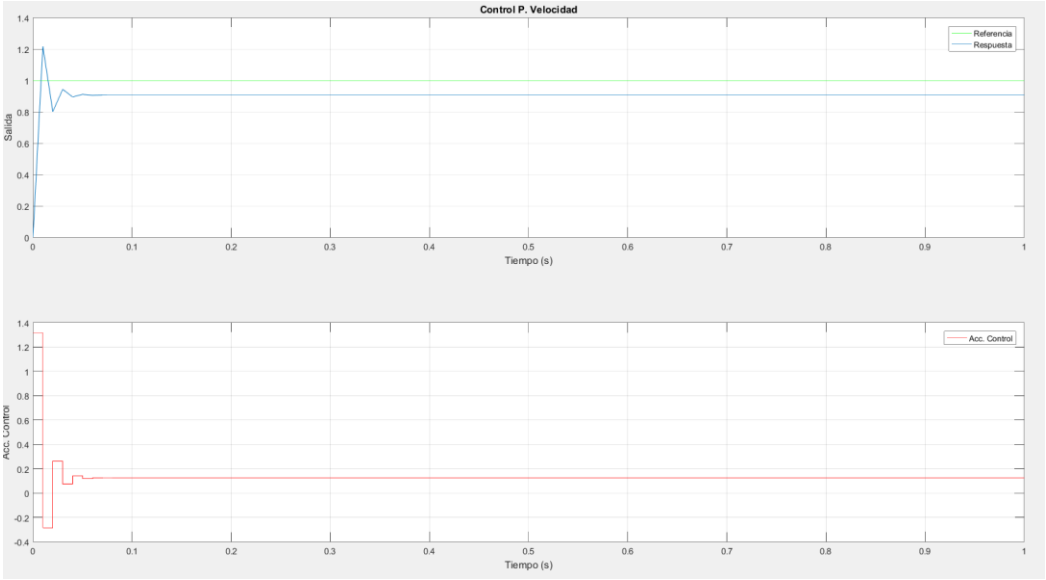


Figura 89: Respuesta del control P de velocidad

Con unos índices de rendimiento de:

$$CIEC = 1.9312$$

$$CIEA = 10.5054$$

Pero, como se puede apreciar, aunque la respuesta se aproxima a la referencia con el error de posición correspondiente a un sistema sin integrador ante una referencia en escalón y el tiempo de establecimiento entra en el deseado, la sobreoscilación, sin embargo es superior a la que se requiere.

Para una referencia de 90º/s:

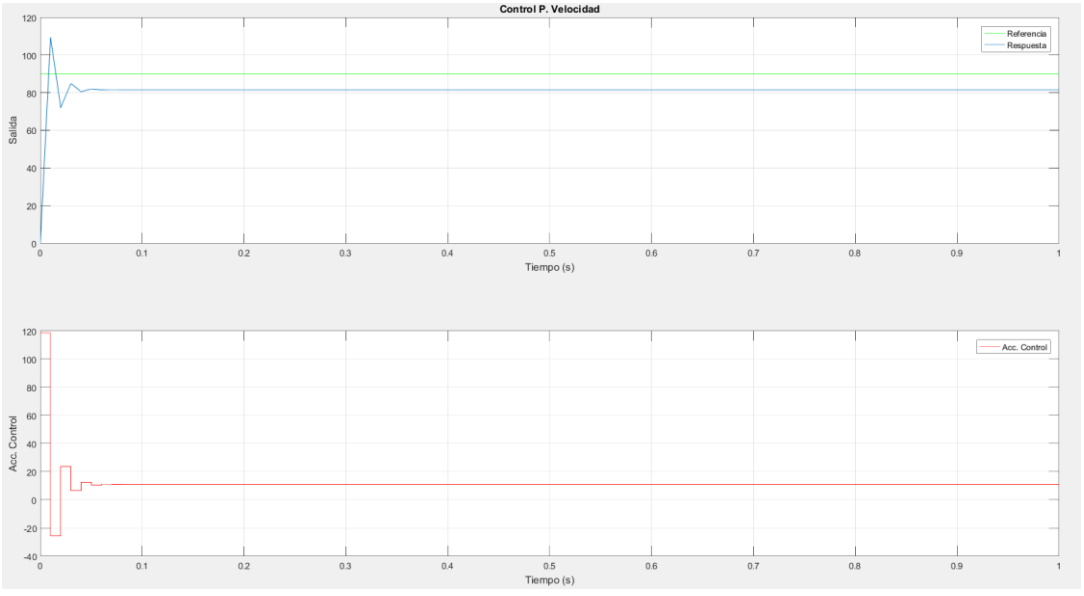


Figura 90: Respuesta del control P de velocidad ante entrada 90º

Donde la acción de control se sale de los valores realizables y se mantiene la sobreoscilación no deseada y el error de posición. La ecuación en diferencias sería:

$$u(k) = 1.3176 \cdot e(k)$$

Para el regulador proporcional-derivativo, con la relación de parámetros de la tabla se tiene:

$$K_{pd} = 1.5811$$

$$T_d = 0.004$$

$$q_0 = 2.2135$$

$$q_1 = -0.6324$$

Con estos valores la respuesta en bucle cerrado es:

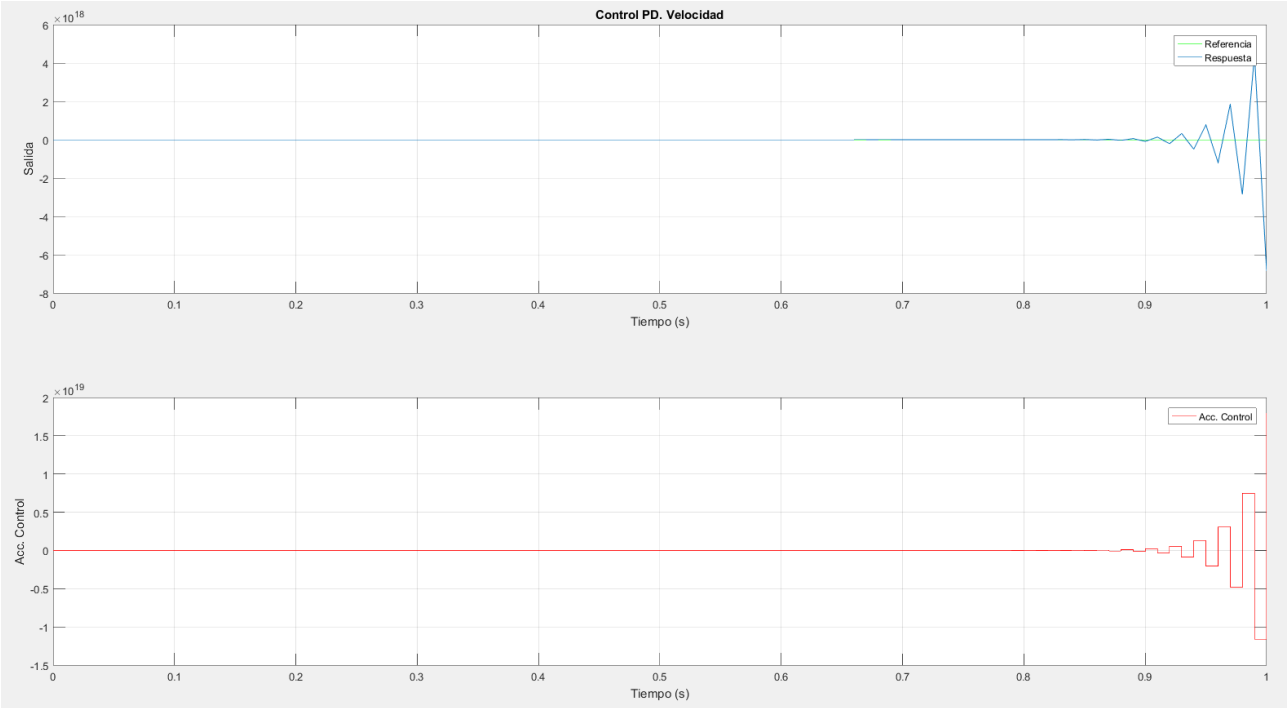


Figura 91: Respuesta del control PD de velocidad

Con unos índices de rendimiento de:

$$CIEC = 8.0320 \cdot 10^{37}$$

$$CIEA = 1.933 \cdot 10^{19}$$

Tal y como se puede apreciar, la acción derivativa acaba desestabilizando el sistema y no es un controlador apropiado para un sistema de primer orden como la velocidad. La ecuación en diferencias sería:

$$u(k) = 2.2135 \cdot e(k) - 0.6324 \cdot e(k-1)$$

Para el regulador proporcional-integral, con la relación de parámetros de la tabla se tiene:

$$K_{pi} = 1.1858$$

$$T_d = 0.0267$$

$$q_0 = 1.1858$$

$$q_1 = -0.7411$$

Con estos valores la respuesta en bucle cerrado es:

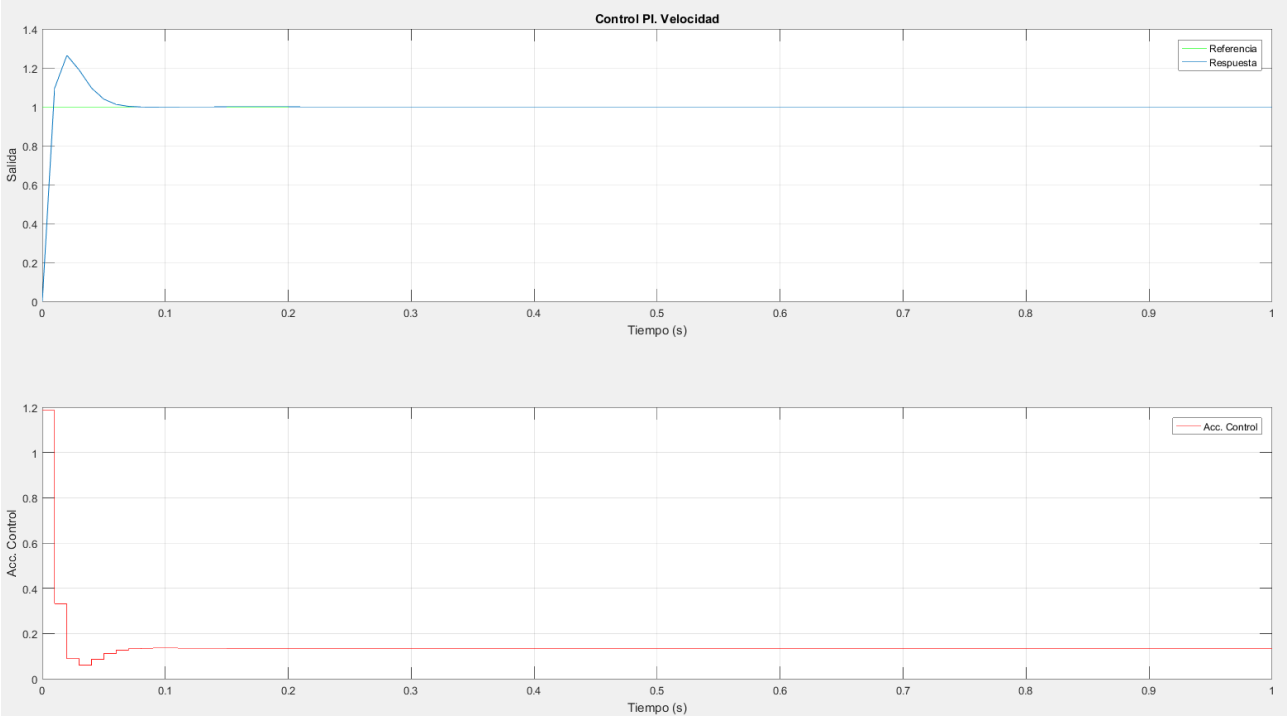


Figura 92: Respuesta del control PI de velocidad

Con unos índices de rendimiento de:

$$CIEC = 1.1258$$

$$CIEA = 1.7016$$

En esta ocasión se alcanza la referencia en el tiempo deseado, gracias al integrador, manteniéndose elevada la sobreoscilación con respecto al diseño. En este punto parece razonable pensar que el criterio de sobreoscilación resulta demasiado restrictivo, sin embargo para un control suave del sistema como es el que se pretende para un movimiento controlado es necesario. Por ello se ajustan manualmente los valores de los parámetros del controlador para situarnos dentro de los límites de nuestro sistema. Con los valores:

$$K_{pi} = 0.1$$

$$T_d = 0.05$$

$$q_0 = 0.1$$

$$q_1 = -0.08$$

Con unos índices de rendimiento de:

$$CIEC = 3.7183 \cdot 10^4$$

$$CIEA = 705.96$$

Tenemos:

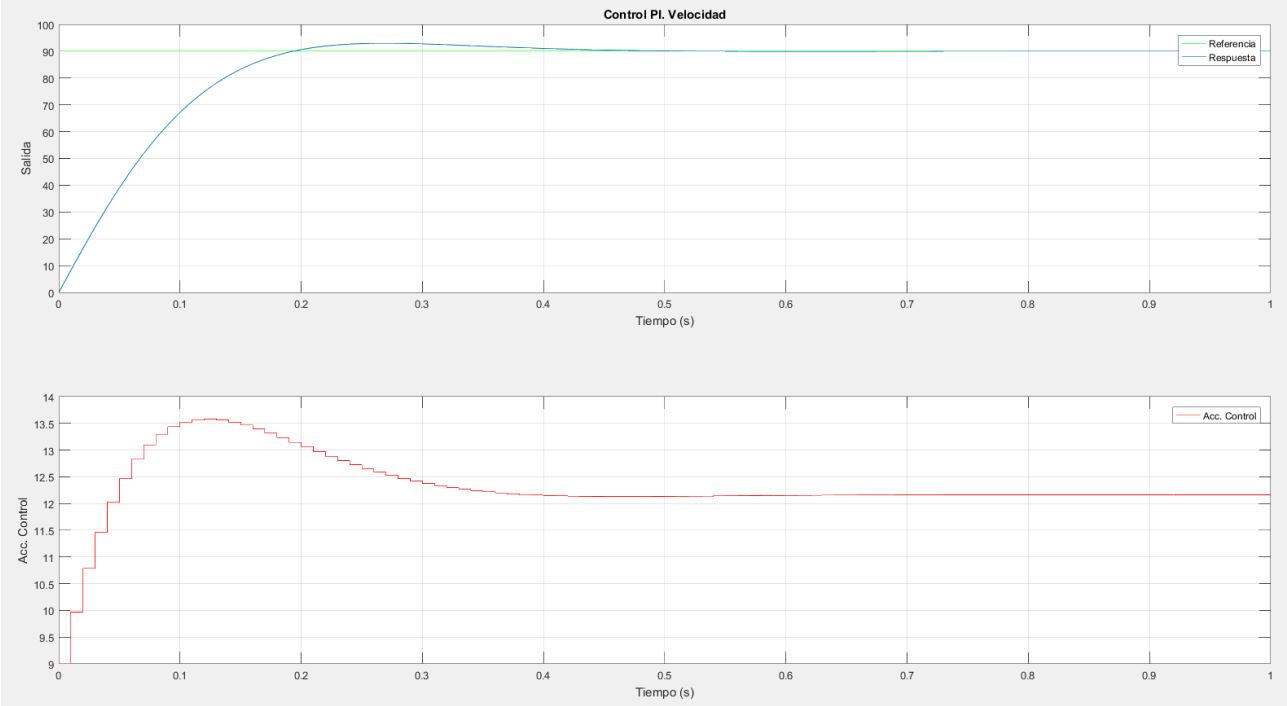


Figura 93: Respuesta del control PI de velocidad ante entrada 90º/s ajustada

Donde cómo podemos apreciar, el tiempo de establecimiento es cuatro veces el deseado pero la acción de control y sobreoscilación cumplen los criterios.

Si quisiéramos alcanzar la máxima velocidad con estas características podríamos ajustar valores de esta manera:

$$K_{pi} = 0.05$$

$$T_d = 0.05$$

$$q_0 = 0.1$$

$$q_1 = -0.08$$

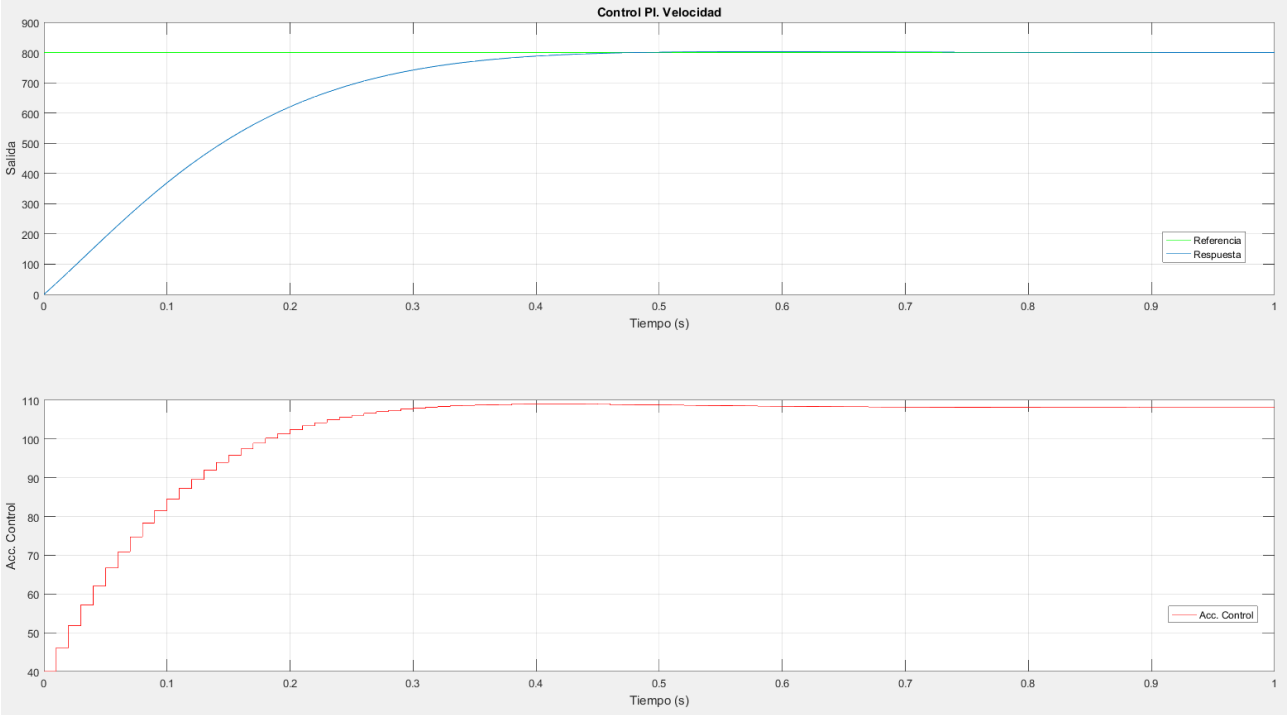


Figura 94: Respuesta del control PI de velocidad ante entrada máxima ajustada

Donde cómo podemos apreciar, el tiempo de establecimiento es cuatro veces el deseado y la acción de control es superior a la máxima. Esto es debido a que para este sistema y con este tipo de controlador es imposible alcanzar las características de diseño requeridas para el seguimiento de una referencia tan restrictiva. Sin embargo, si se relajan las especificaciones de tiempo de establecimiento o se sigue una referencia menos restrictiva como una velocidad de 720 º/s o de 360º/s:

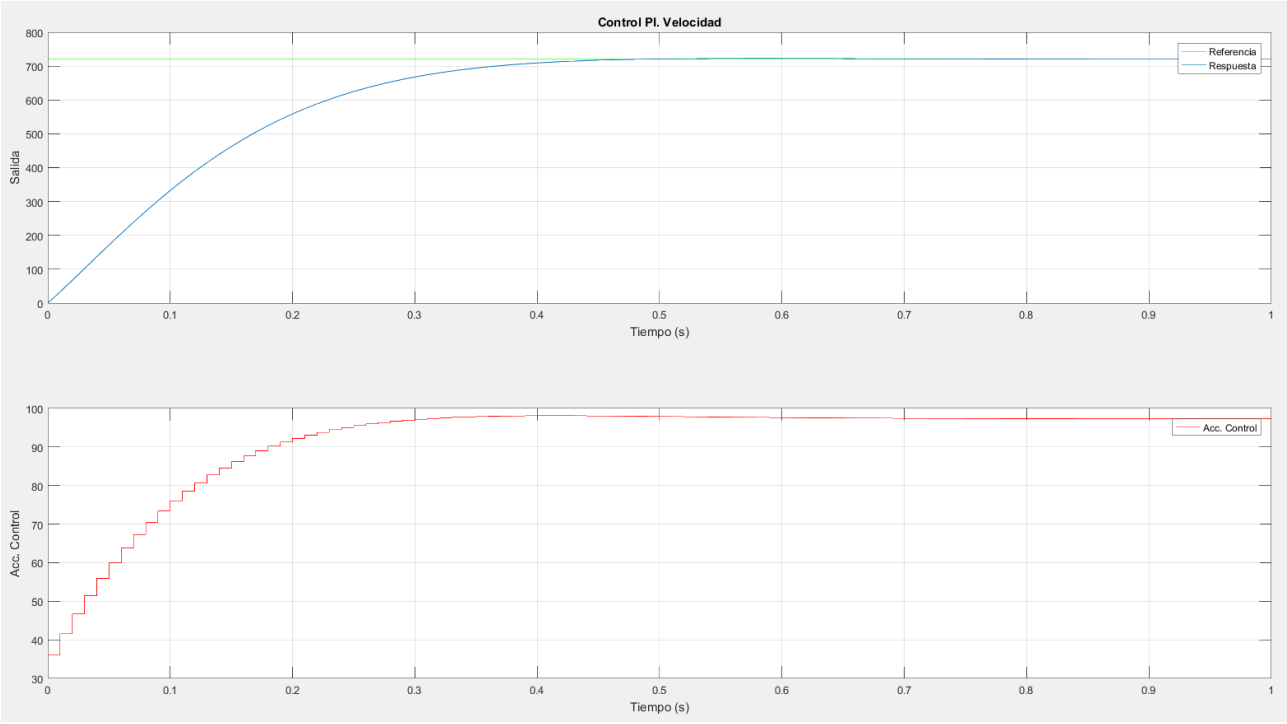


Figura 95: Respuesta del control PI de velocidad ante entrada 720º/s ajustada

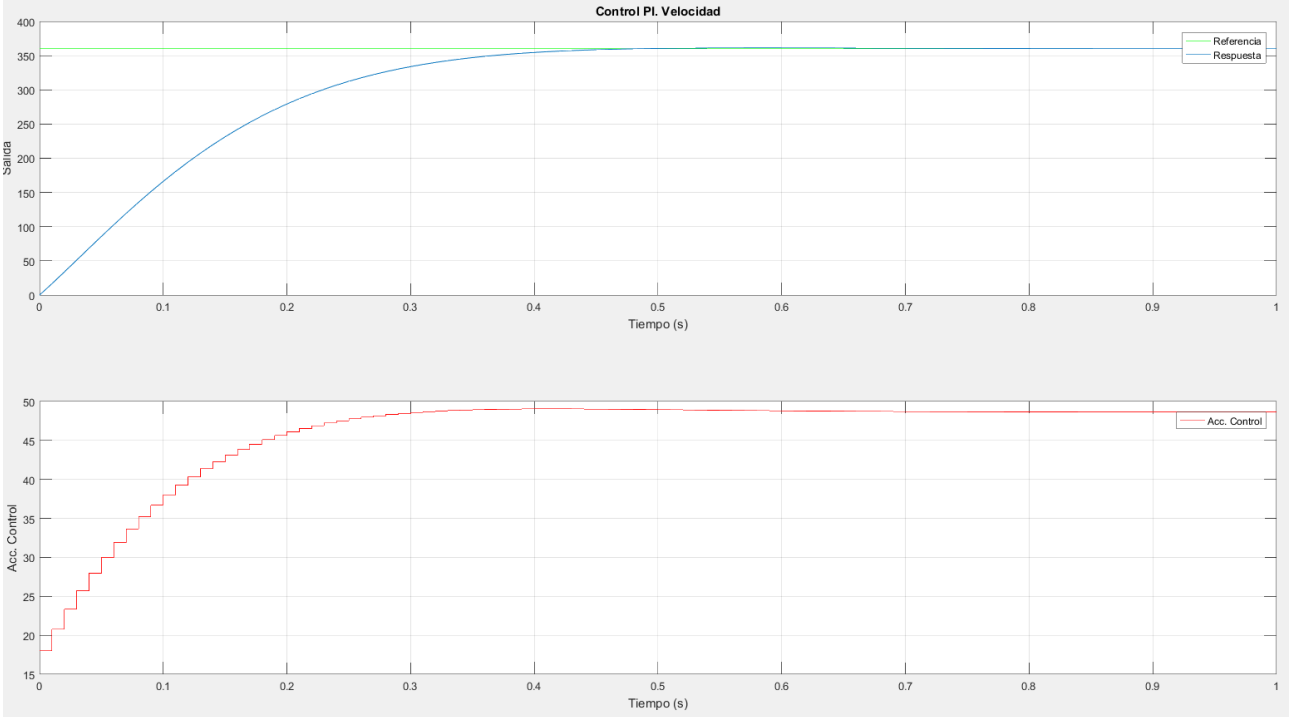


Figura 96: Respuesta del control PI de velocidad ante entrada 360º/s ajustada

Se tendría, por tanto, una aproximación lo bastante buena para alcanzar una trayectoria que solicite cambios de aceleración elevados con este regulador.

La ecuación en diferencias para el controlador diseñado con referencia 90 º/s sería:

$$u(k) = 0.01 \cdot e(k) - 0.08 \cdot e(k-1) + u(k-1)$$

Ecuación 50. Ecuación en diferencias del PI a implementar

Por último, para el regulador proporcional-integral-derivativo, con la relación de parámetros de la tabla se tiene:

$$K_{pid} = 1.5811$$

$$T_i = 0.0160$$

$$T_d = 0.004$$

$$q_0 = 2.21$$

$$q_1 = -1.86$$

$$q_2 = 0.6324$$

Con estos valores la respuesta en bucle cerrado es:

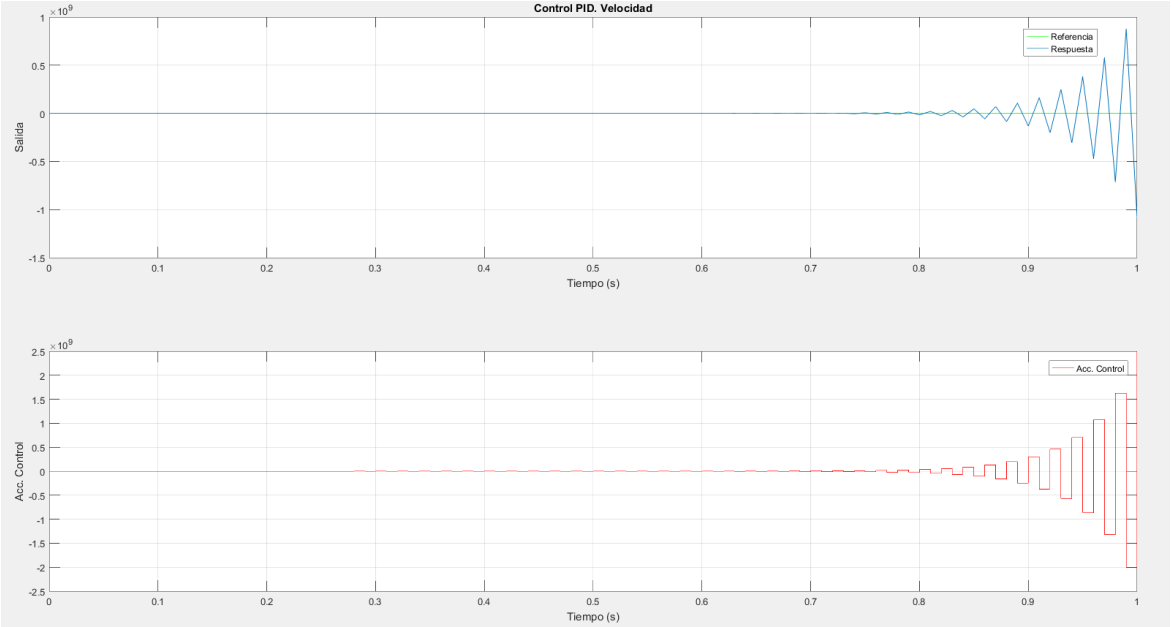


Figura 97: Respuesta del control PID de velocidad

Con unos índices de rendimiento de:

$$CIEC = 3.437 \cdot 10^{18}$$

$$CIEA = 5.725 \cdot 10^{9}$$

De nuevo, el sistema se hace inestable por la acción derivativa. No se realizará el ajuste manual de los parámetros. La ecuación en diferencias sería:

$$u(k) = 2.21 \cdot e(k) - 1.86 \cdot e(k-1) + 0.6324 \cdot e(k-2) + u(k-1)$$

Resulta, tras la simulación de todos los controles, el PI como control más adecuado, presentando menor error y con mejores características para el control de la velocidad.

Para el modelo de posición, con el ajuste de respuesta sostenida se tiene:

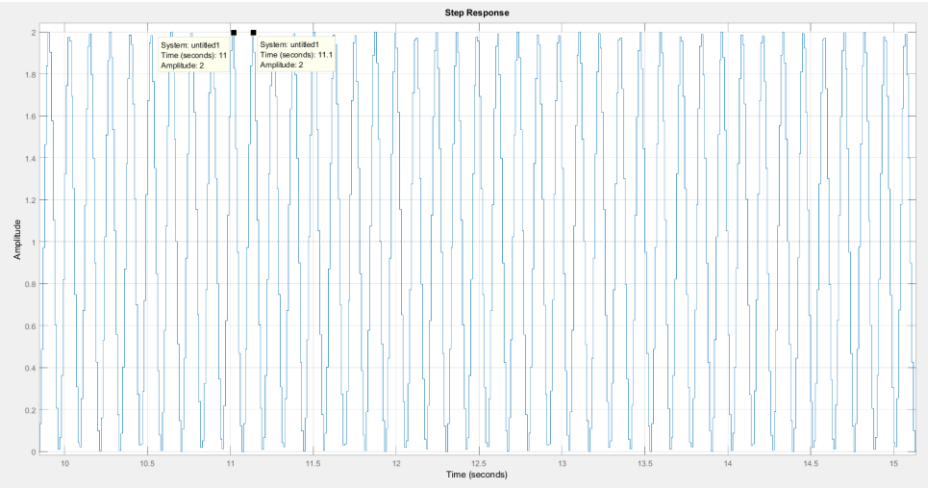


Figura 98: Respuesta sostenida del modelo

Donde:

$$K_u = 27.056$$

$$T_c = 0.1$$

Podemos ahora obtener los parámetros del control P, PD, PI y PID y comprobar su respuesta mediante simulación a partir del mismo esquema de *Simulink* y su variante con la saturación de la acción de control.

Se recuerda de nuevo nuestros parámetros de respuesta deseados (tiempo de establecimiento inferior a 0.1s y sobreoscilación inferior al 5%). En este caso como referencia se tomará un valor de 90º para comprobar la respuesta de las distintas configuraciones con el método de la respuesta sostenida, siendo este algo superior al máximo que producirá nuestro generador en un determinado instante pero permite la comprobación visual sencilla en la plataforma de ensayos. Teniendo esto en cuenta, los índices de rendimiento como indicador del error y la limitación de la acción de control se procede al estudio de estos controladores para la posición.

Así, para el regulador proporcional, con la relación de parámetros de la tabla se tiene:

$$K_p = 13.5280$$

Con este valor, la respuesta del sistema en bucle cerrado es:

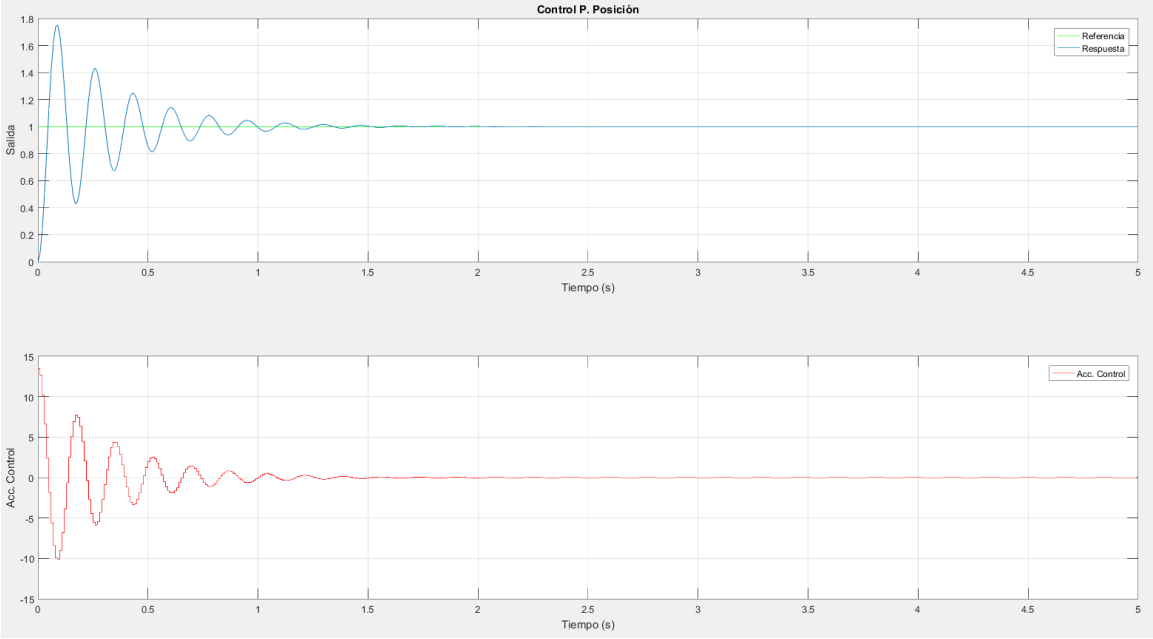


Figura 99: Respuesta del control P de posición

Con unos índices de rendimiento de:

$$CIEC = 8.4946$$

$$CIEA = 20.4647$$

Se puede comprobar que sobreoscilación y tiempo de establecimiento se encuentran fuera de

los límites de diseño. Ante una entrada de 90º se tiene:

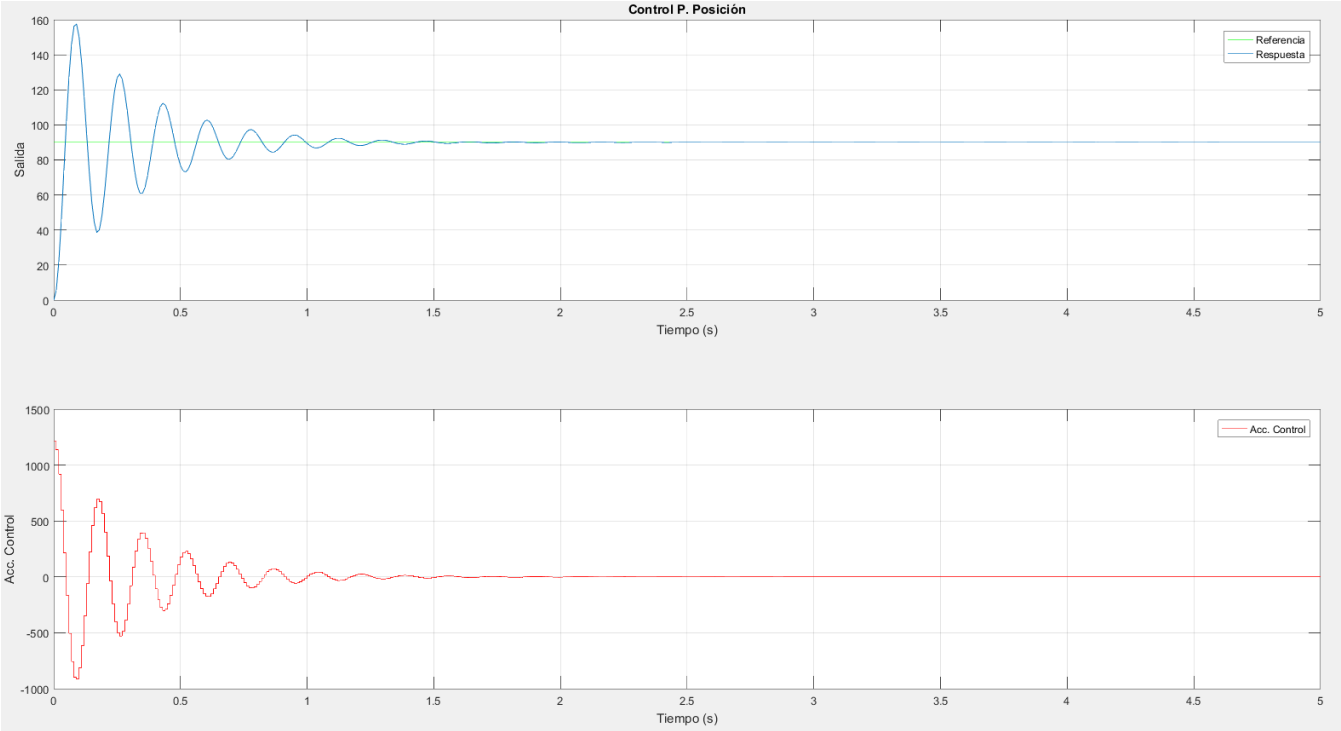


Figura 100: Respuesta del control P de posición ante entrada de 90º

Lo que también deja fuera de los límites la acción de control. Se podría ajustar el valor del parámetro del controlador para incluirlo dentro de los valores aceptables, pero antes estudiaremos el resto de casos. La ecuación en diferencias tendría esta forma:

$$u(k) = 13.528 \cdot e(k)$$

Para el regulador proporcional-derivativo, con la relación de parámetros de la tabla se tiene:

$$K_{pd} = 16.2336$$

$$T_d = 0.0125$$

$$q_0 = 36.5256$$

$$q_1 = -20.2920$$

Con estos valores la respuesta en bucle cerrado es:

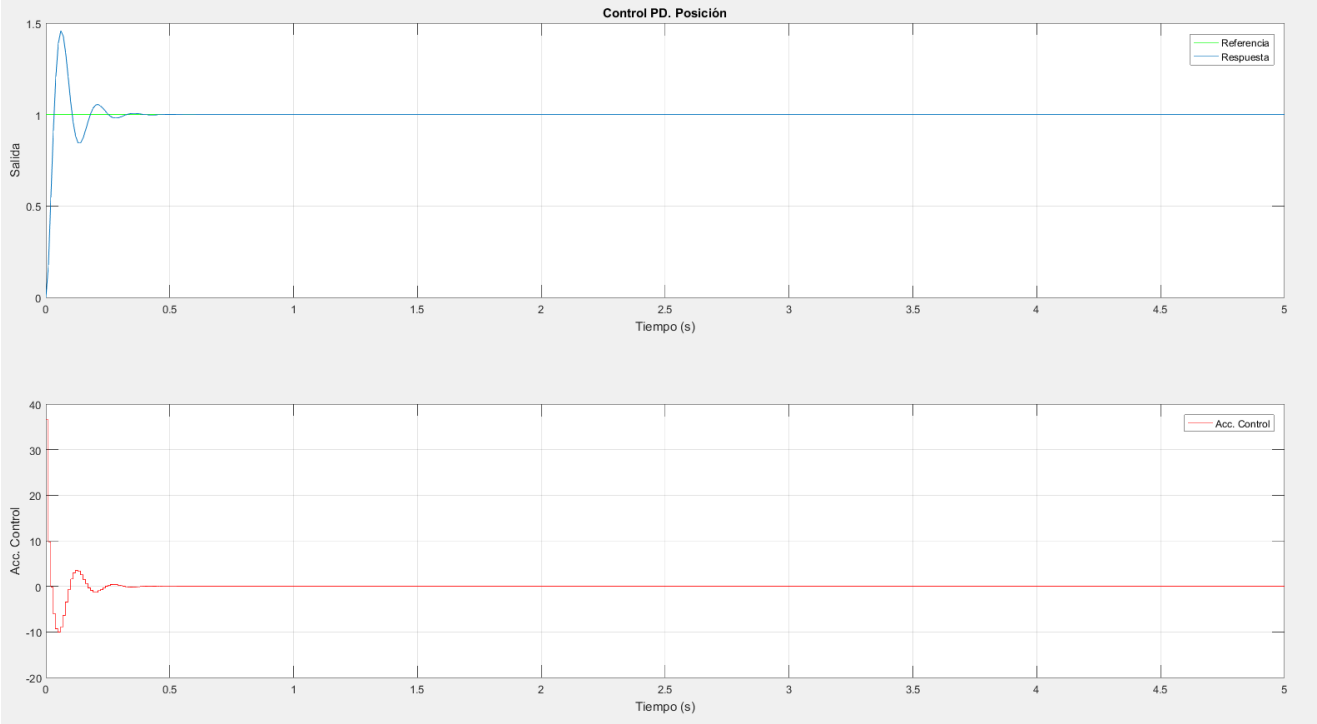


Figura 101: Respuesta del control PD de posición

Con unos índices de rendimiento de:

$$CIEC = 2.7407$$

$$CIEA = 5.5625$$

Tal y como se puede apreciar, la respuesta del sistema mejora notablemente respecto a la equivalente del control proporcional en sobreoscilación, pero a costa de una mayor acción de control. La ecuación en diferencias tendría esta forma:

$$u(k) = 36.5256 \cdot e(k) - 20.292 \cdot e(k-1)$$

Ecuación 51. Ecuación en diferencias del PD a implementar

Pero ante una entrada de 90º:

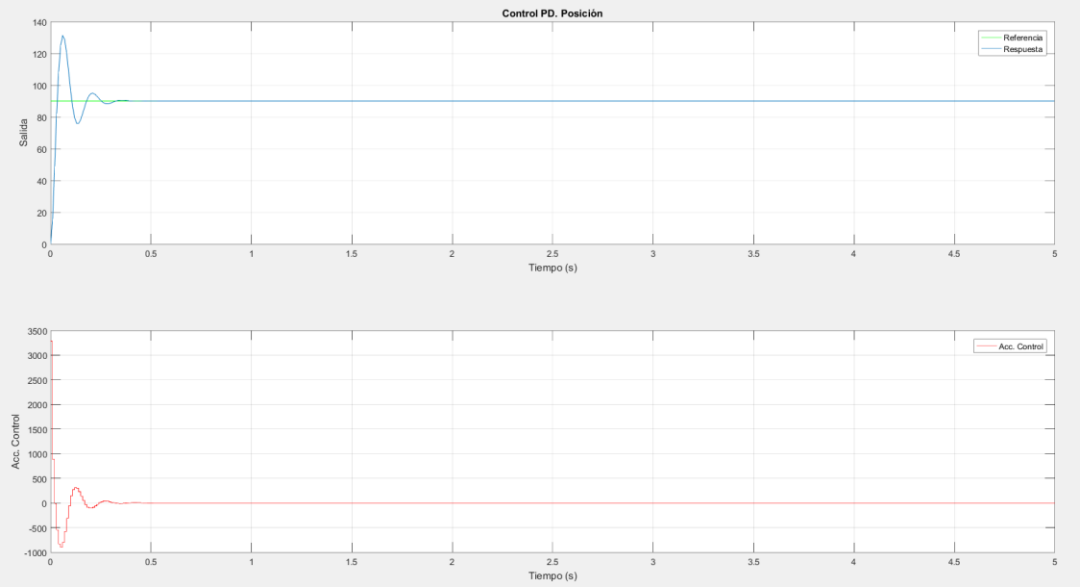


Figura 102: Respuesta del control PD de posición ante entrada de 90º

La acción de control es muy superior a la que somos capaces de administrar. Se ajustará el valor de los parámetros del controlador más adelante si este resulta el que mejores características de control presenta.

Para el regulador proporcional-integral, con la relación de parámetros de la tabla se tiene:

$$K_{pi} = 12.1752$$

$$T_d = 0.0833$$

$$q_0 = 12.1752$$

$$q_1 = -10.7142$$

Con estos valores la respuesta en bucle cerrado es:

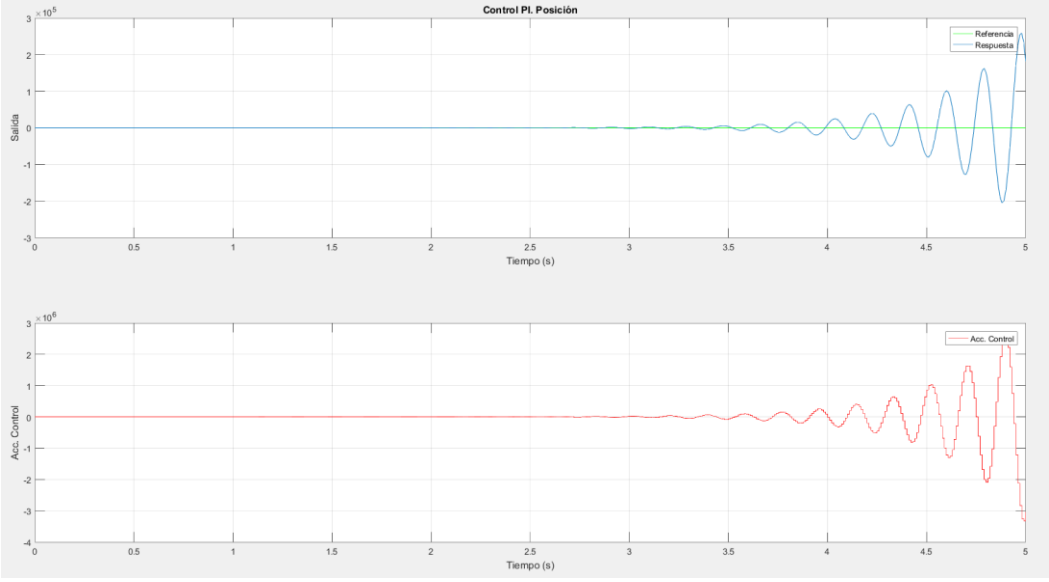


Figura 103: Respuesta del control PI de posición

Con unos índices de rendimiento de:

$$CIEC = 8.355 \cdot 10^{15}$$

$$CIEA = 7.298 \cdot 10^{8}$$

En esta ocasión y con los valores obtenidos en la tabla la respuesta del sistema se hace inestable, siendo el equivalente al derivador en el modelo de velocidad, desestimando este controlador como adecuado para el proceso a controlar.

Por último, para el regulador proporcional-integral-derivativo, con la relación de parámetros de la tabla se tiene:

$$K_{pid} = 16.2336$$

$$T_i = 0.0833$$

$$T_d = 0.0125$$

$$q_0 = 36.5256$$

$$q_1 = -54.8696$$

$$q_2 = 20.2920$$

Con estos valores la respuesta en bucle cerrado es:

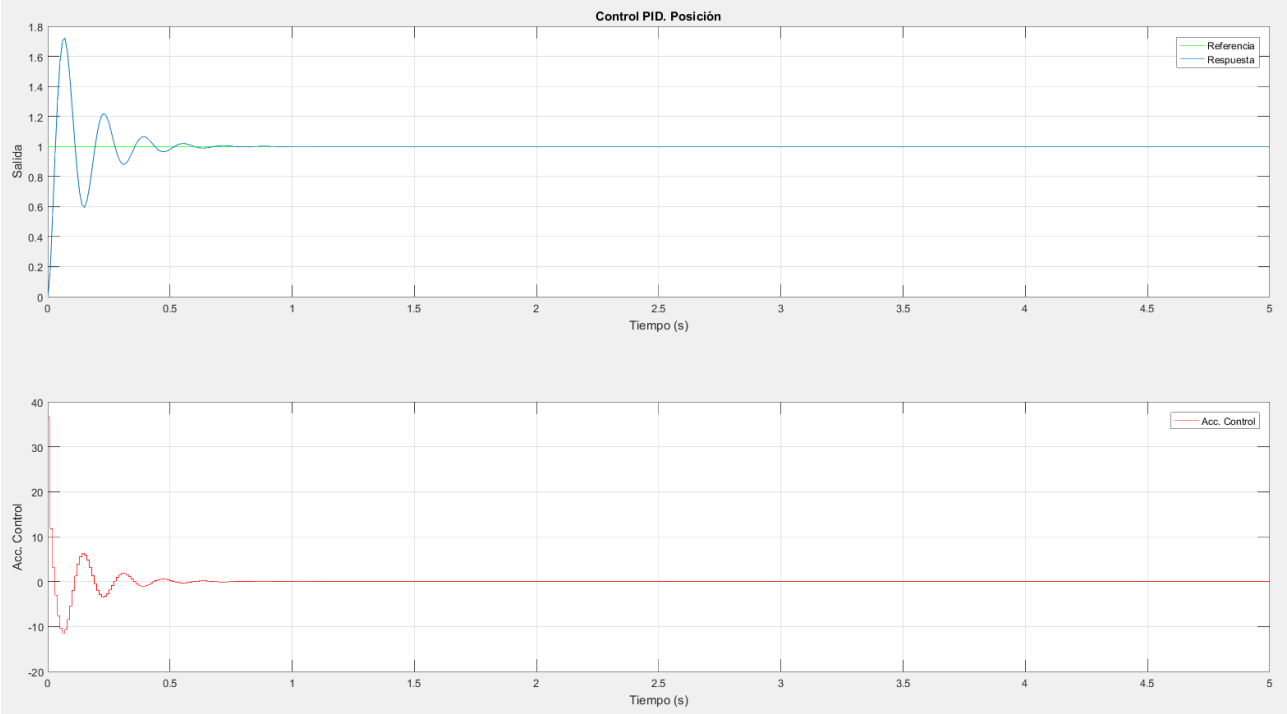


Figura 104: Respuesta del control PID de posición

Con unos índices de rendimiento de:

$$CIEC = 4.9300$$

$$CIEA = 10.5853$$

Podemos apreciar en esta respuesta un peor comportamiento en todos los aspectos respecto al control PD. La ecuación en diferencias sería:

$$u(k) = 36.5256 \cdot e(k) - 54.8696 \cdot e(k-1) + 20.2920 \cdot e(k-2) + u(k-1)$$

Resulta, tras la simulación de todos los controles, el PD control más adecuado y con mejores características para el control de la posición. No obstante requiere de un ajuste para adecuar la acción de control a los límites de nuestro sistema.

Realizando el ajuste manual, para el controlador PD tenemos los siguientes valores:

$$K_{pd} = 0.05$$

$$T_d = 0.0125$$

$$q_0 = 1.125$$

$$q_1 = -0.625$$

$$CIEC = 1.33 \cdot 10^5$$

$$CIEA = 2.38 \cdot 10^3$$

Consiguiendo la siguiente respuesta:

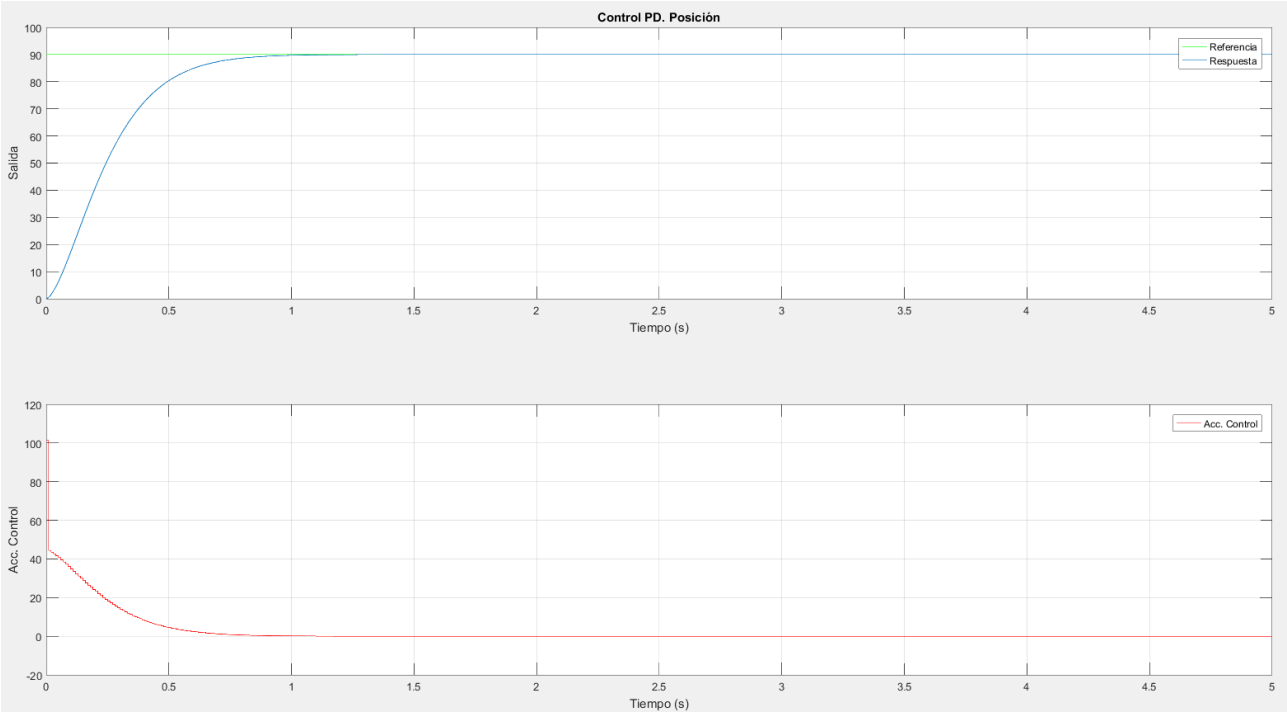


Figura 105: Respuesta del control PD de posición ante entrada de 90º ajustada

Consiguiendo una excelente respuesta salvo por el tiempo de establecimiento.

Como se indica posteriormente en el punto 5, el máximo desplazamiento en grados que podrá efectuar el motor para desplazar el carro a lo largo de nuestro eje de la plataforma de ensayos. Este valor es de 3750 º. Si establecemos este valor como referencia y ajustamos parámetros para conseguir que se encuentre dentro de los límites de acción de control, se puede tener una idea aproximada de lo que costaría alcanzar una referencia en escalón de nuestro desplazamiento máximo sin el perfil de movimiento que posteriormente desarrollaremos.

Conseguimos para el controlador PD la siguiente respuesta:

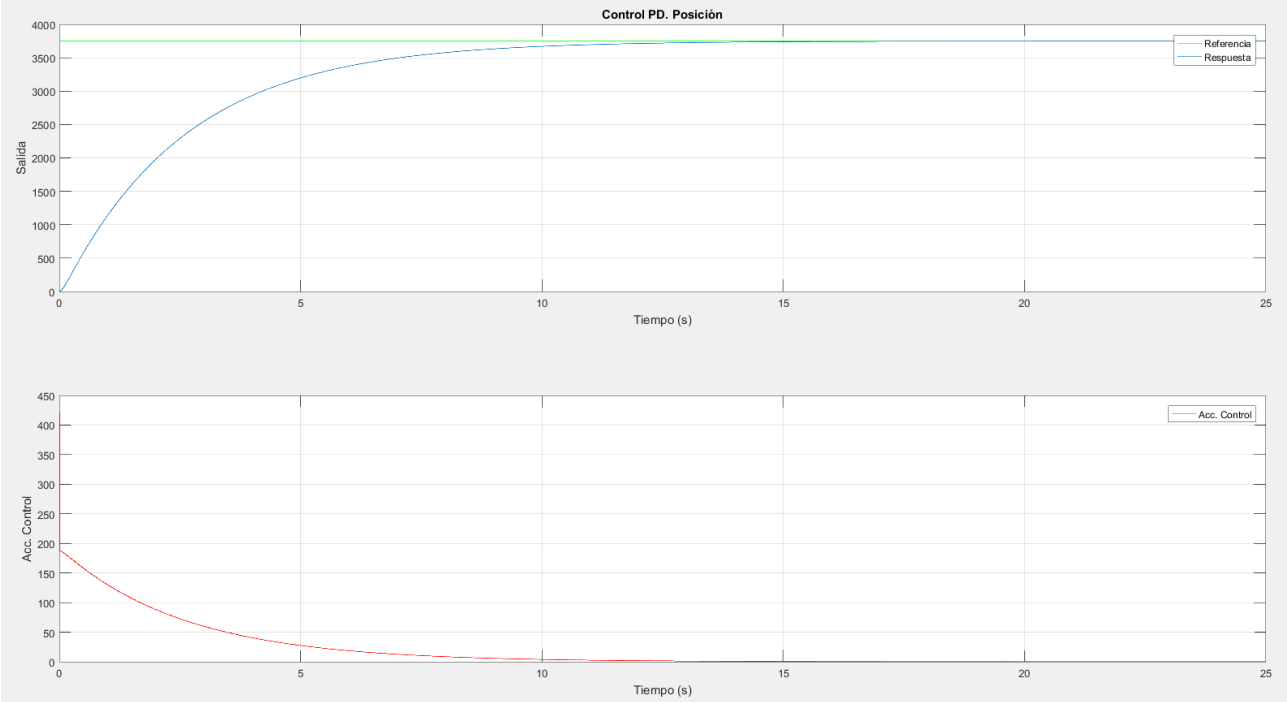


Figura 106: Respuesta del control PD de posición ante entrada de 3750 ajustada

Que alcanza la referencia tras 13 s necesitando una entrada inicial que nuestro sistema no puede proporcionar.

5.3.3. Resultados. Implementación del control y realización de movimientos punto a punto

Tras el diseño y ajuste de los controles de posición y velocidad vamos a proceder a la implementación de los mismos en nuestro sistema. En el anexo [ANEXO A1.3-4] se encuentra el código relativo a la implementación práctica de estos reguladores en el microcontrolador y el posterior tratamiento de los datos obtenidos para la comparación de las respuestas.

Para el control de velocidad mediante PI, con referencias de 180 y 360 º/s se consigue la siguiente respuesta:

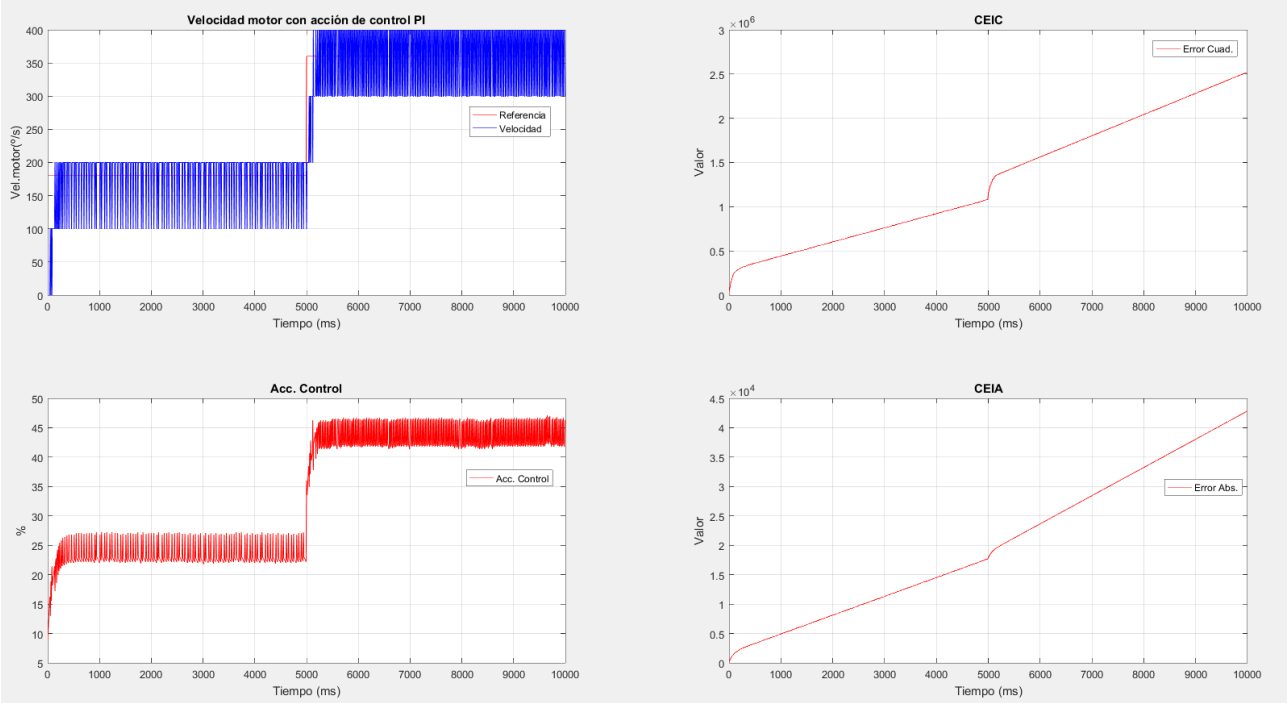


Figura 107: Resultado de implementación del control PI de velocidad ante un escalón de 180 º/s a 360º/s

Para el seguimiento de referencias de 360 y 800 º/s se consigue:

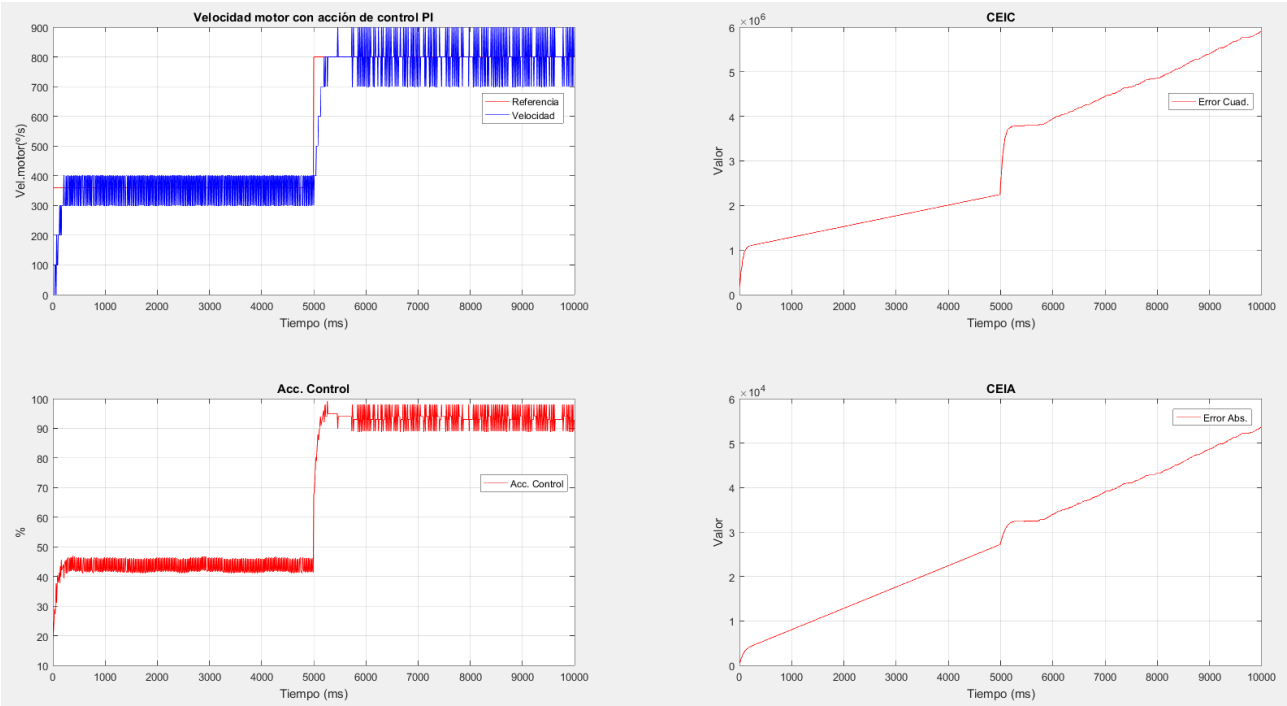


Figura 108: Resultado de implementación del control PI de velocidad ante un escalón de 360 º/s a 800º/s

Donde se aprecia como nuestro sistema sigue la referencia tal y como se conseguía en la simulación.

Para el control de posición mediante PD, utilizaremos como referencia una rampa de valor final el máximo movimiento que puede realizar el carro en el eje del prototipo de ensayos en un tiempo de 15s. Se consigue la siguiente respuesta:

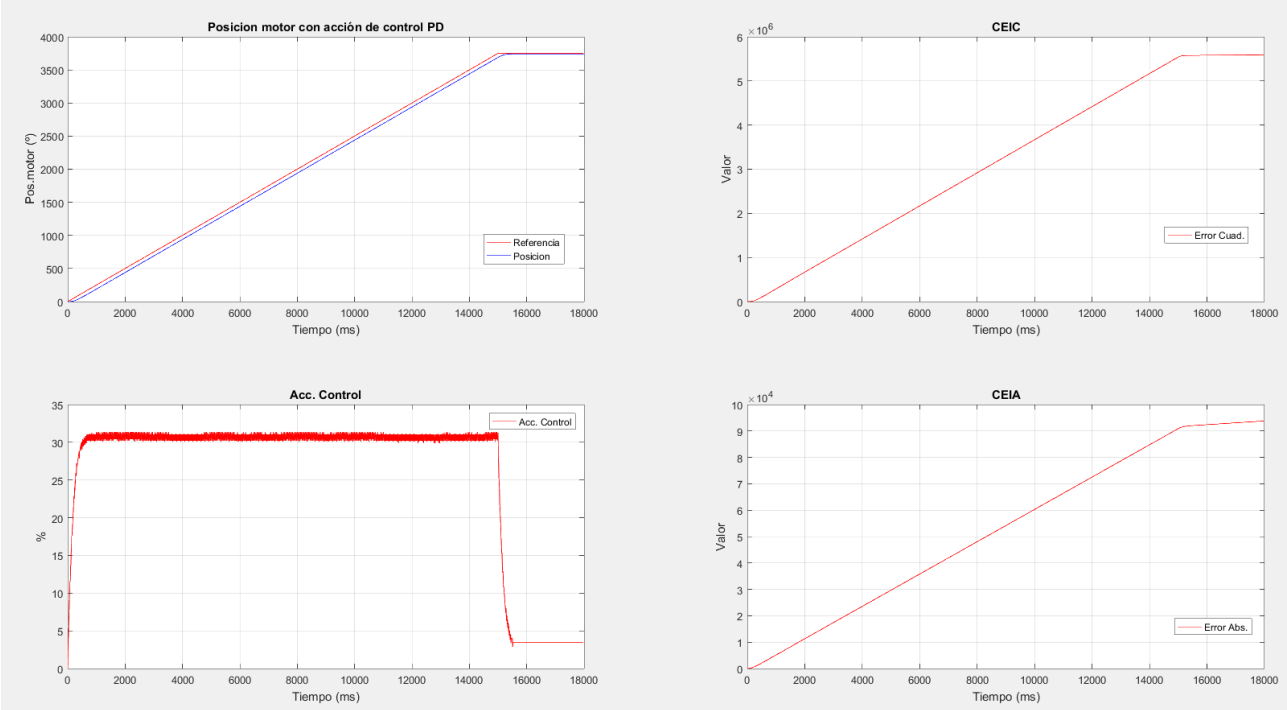


Figura 109: Resultado de implementación del control PD de posición con perfil de movimiento en rampa

Para una rampa de 8 s se consigue:

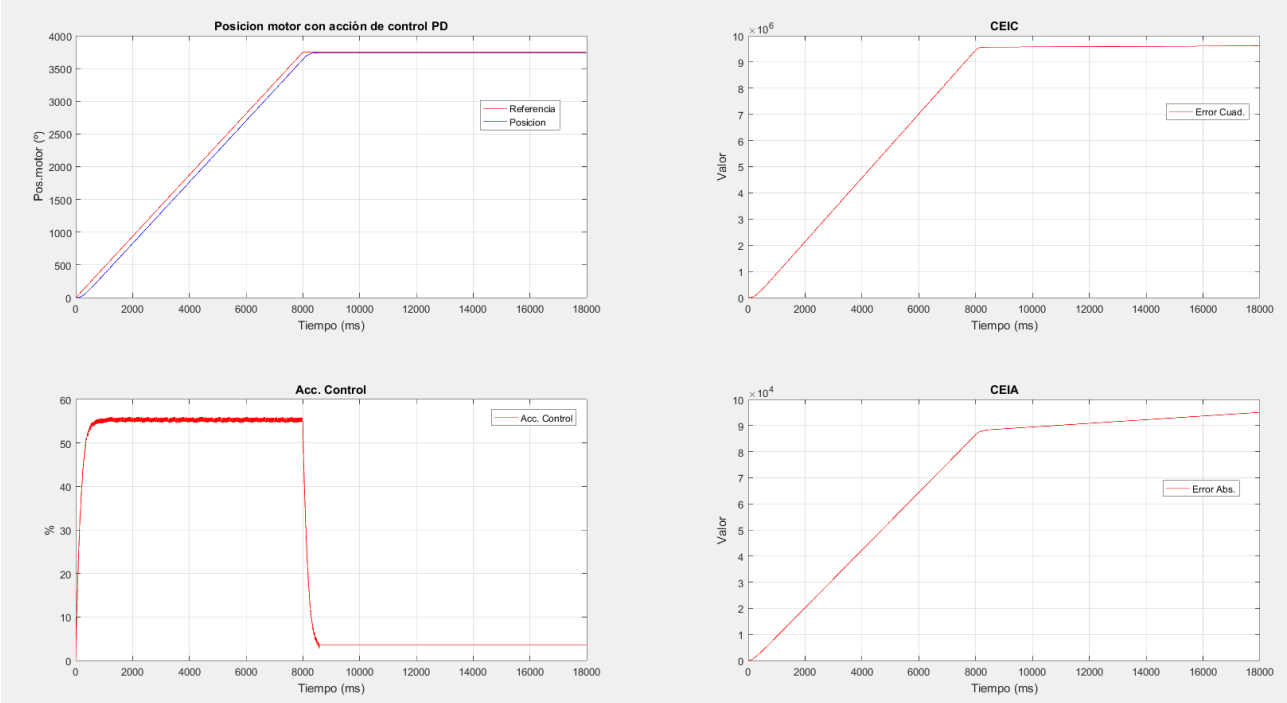


Figura 110: Resultado de implementación del control PD de posición con perfil de movimiento en rampa más restrictiva

Donde se aprecia como nuestro sistema sigue la referencia tal y como se conseguía en la simulación.

Si observamos las respuestas al movimiento utilizando una referencia más suave a partir de nuestro generador de trayectorias tenemos para el control PD y un perfil de movimiento trapezoidal:

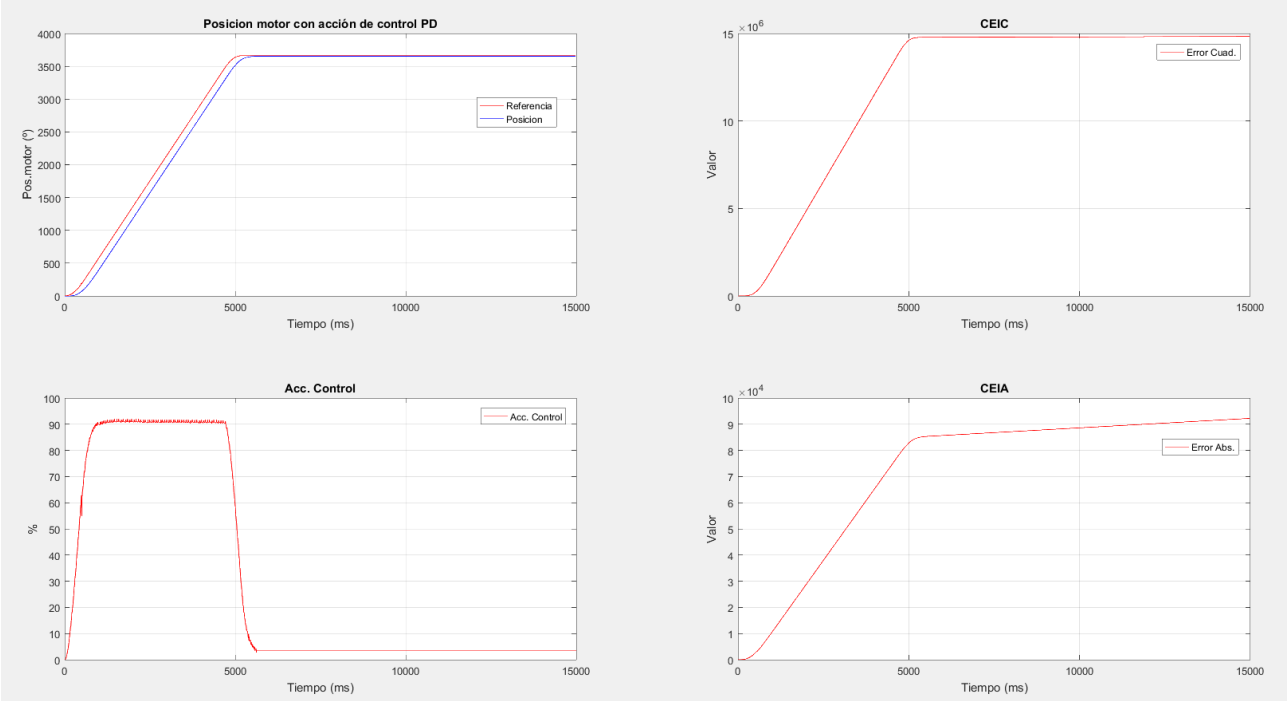


Figura 109: Resultado de implementación del control PD de posición con perfil de movimiento trapezoidal

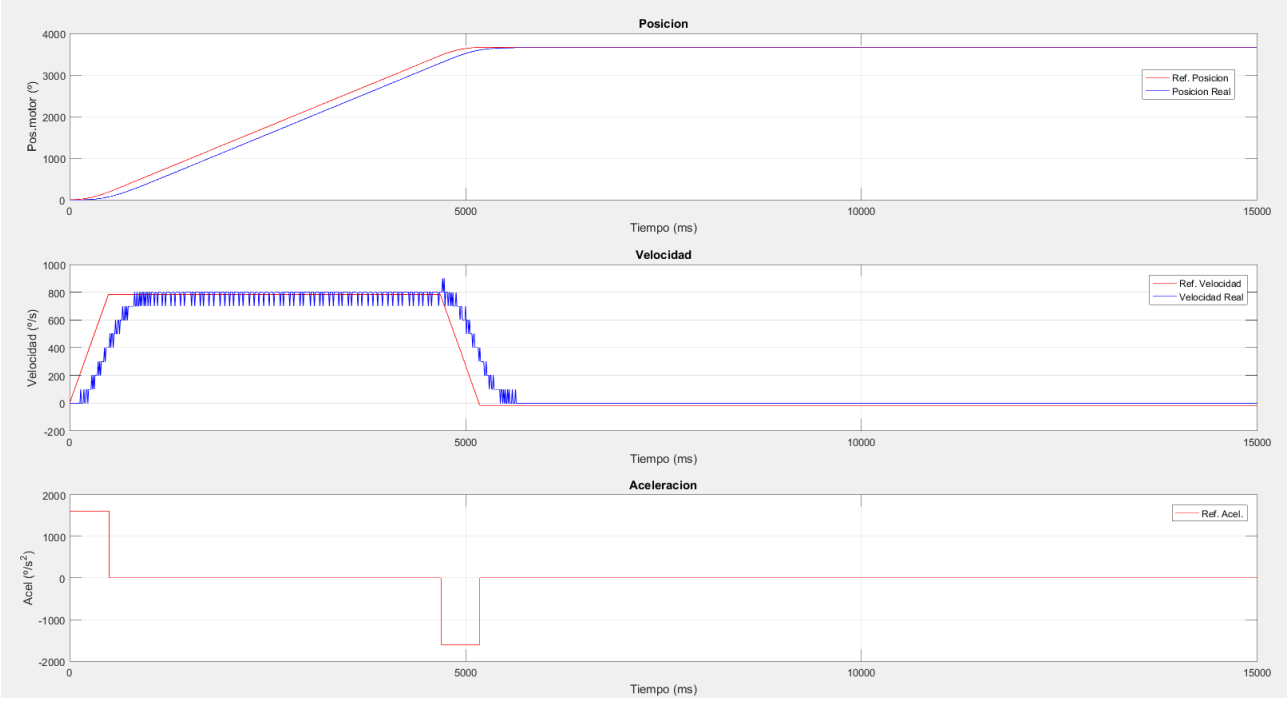


Figura 110: Resultado de implementación del control PD de posición con perfil de movimiento trapezoidal

Para el mismo control y un perfil de movimiento de curva en S de tercer orden:

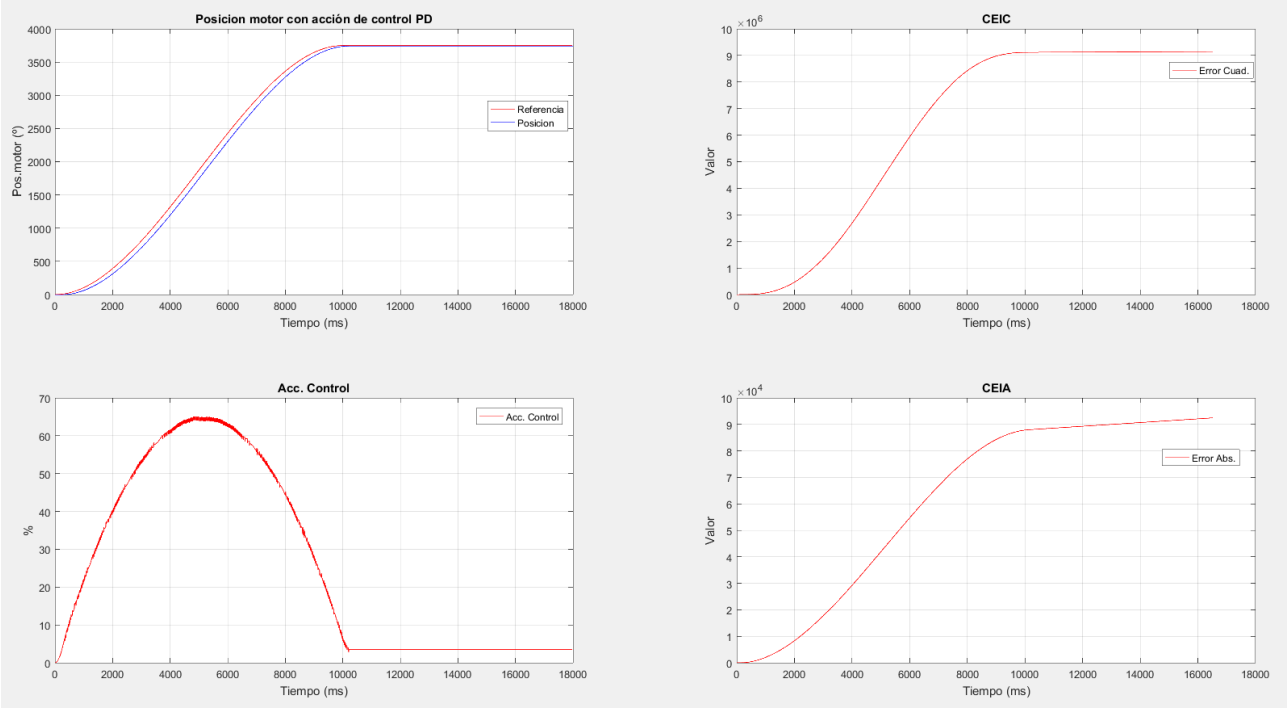


Figura 111: Resultado de implementación del control PD de posición con perfil de movimiento de tercer orden

Puede apreciarse la mejora en la respuesta y la ejecución del movimiento. Podemos graficar la velocidad también para comprobar cómo, pese a realizar el control de la posición directamente, se sigue la señal de velocidad que genera la referencia de posición:

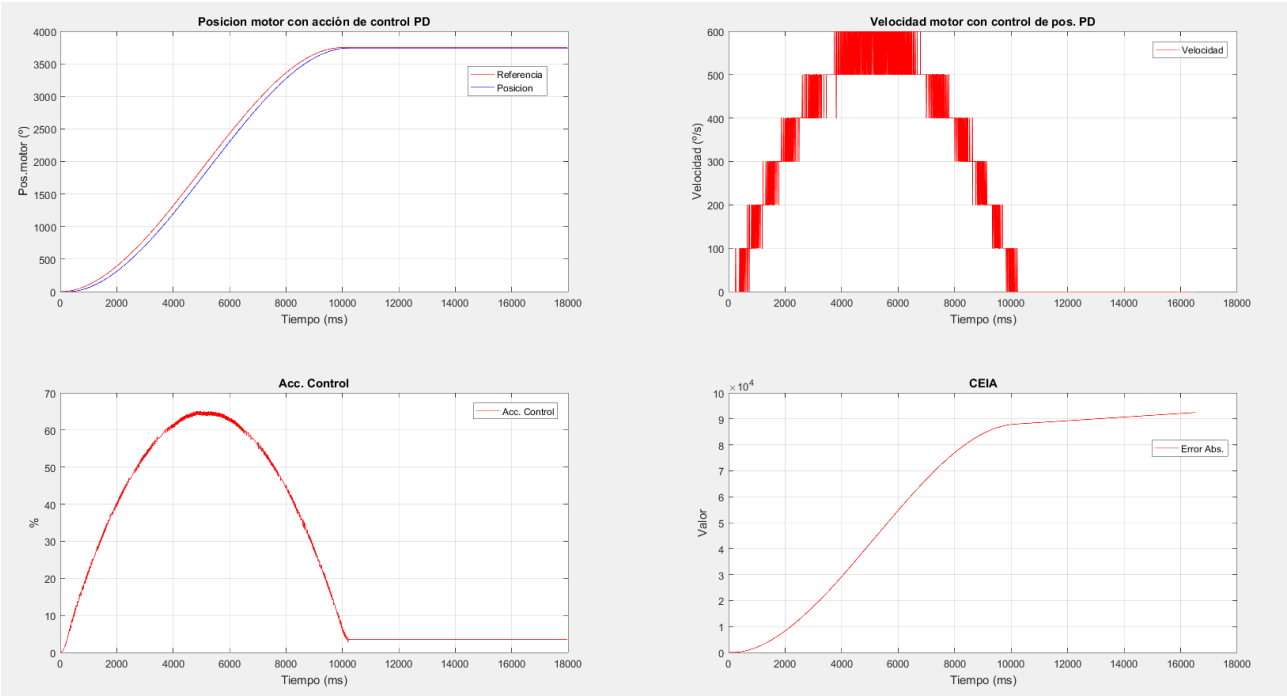


Figura 112: Resultado de implementación del control PD de posición con perfil de movimiento de tercer orden

Podemos comprobar el efecto de una perturbación introducida una vez alcanzada la referencia en el controlador PD al desplazar con la mano el carro cuando ha acabado el movimiento. Si, como se espera, nuestro control realiza su función correctamente anulará el error introducido por

esta perturbación:

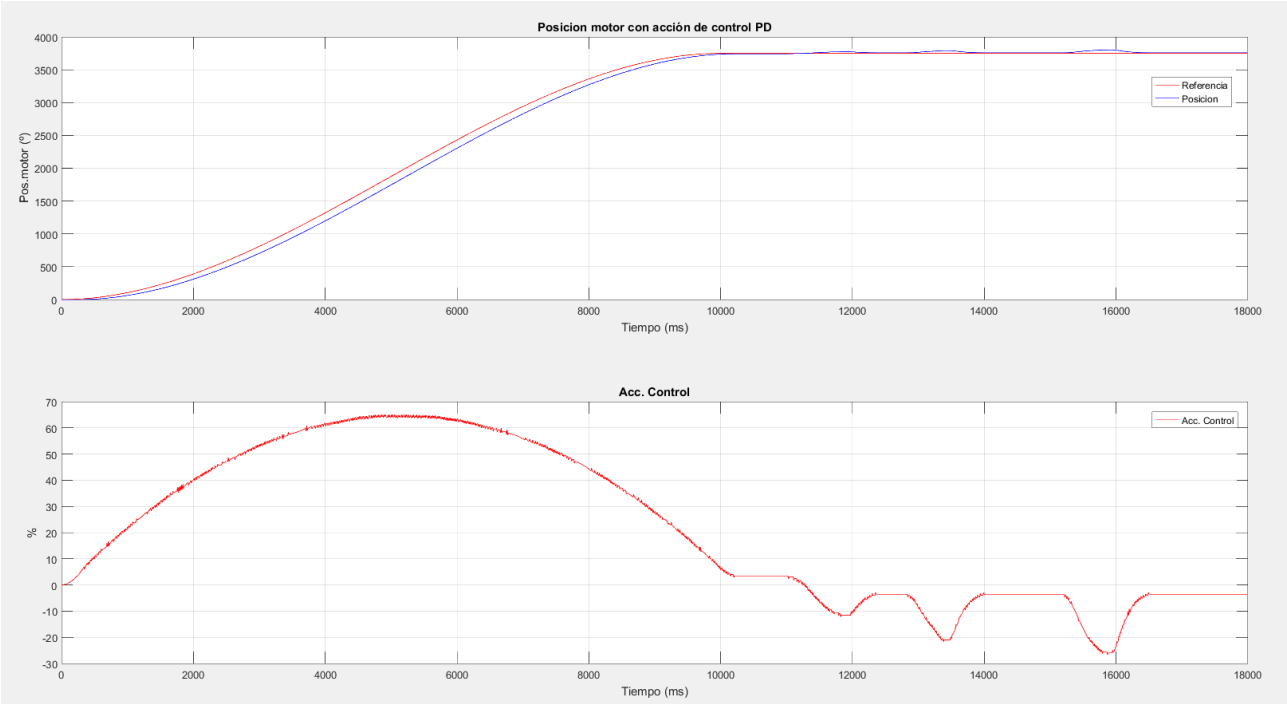


Figura 113: Resultado de implementación del control PD de posición con perfil de movimiento de tercer orden y perturbaciones

Tal y como se puede apreciar, el controlador diseñado e implementado cuando se producen las tres perturbaciones actúa en consecuencia para compensarlas.

Si realizamos el mismo ensayo para un perfil trapezoidal, pero realizando el control para la velocidad con nuestro regulador PI, tenemos:

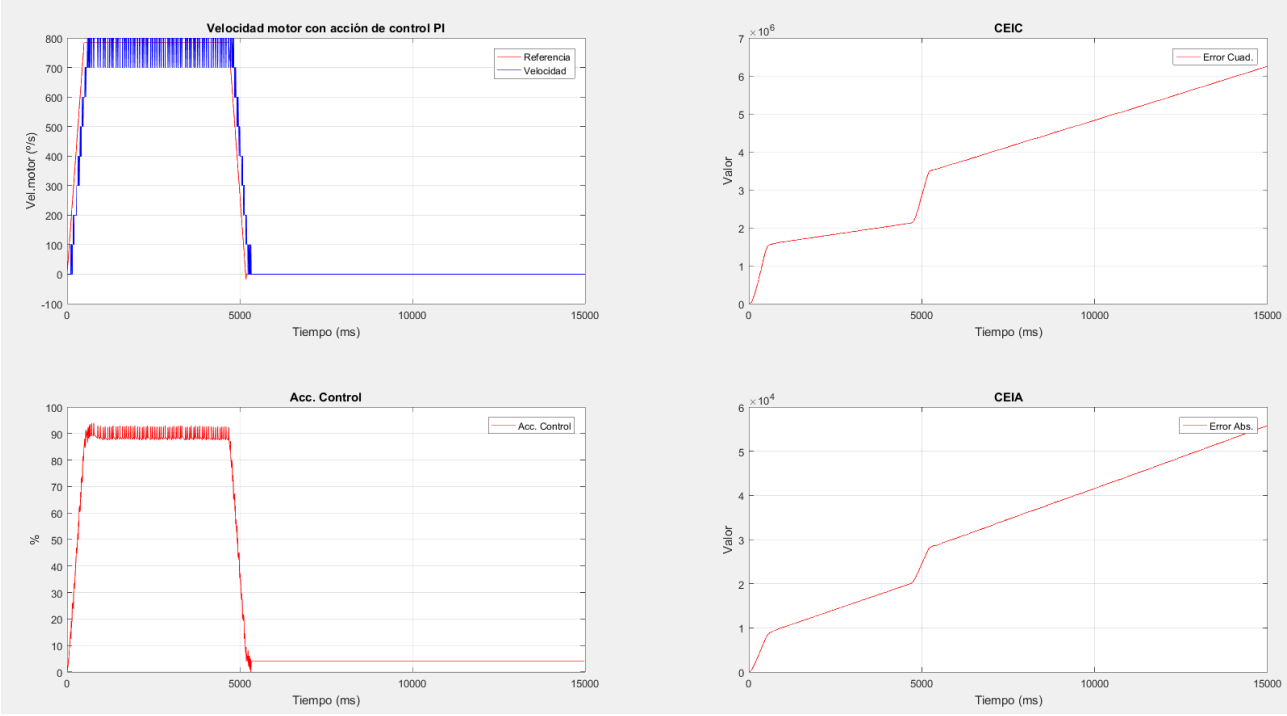


Figura 114: Resultado de implementación del control PI de velocidad con perfil de movimiento trapezoidal

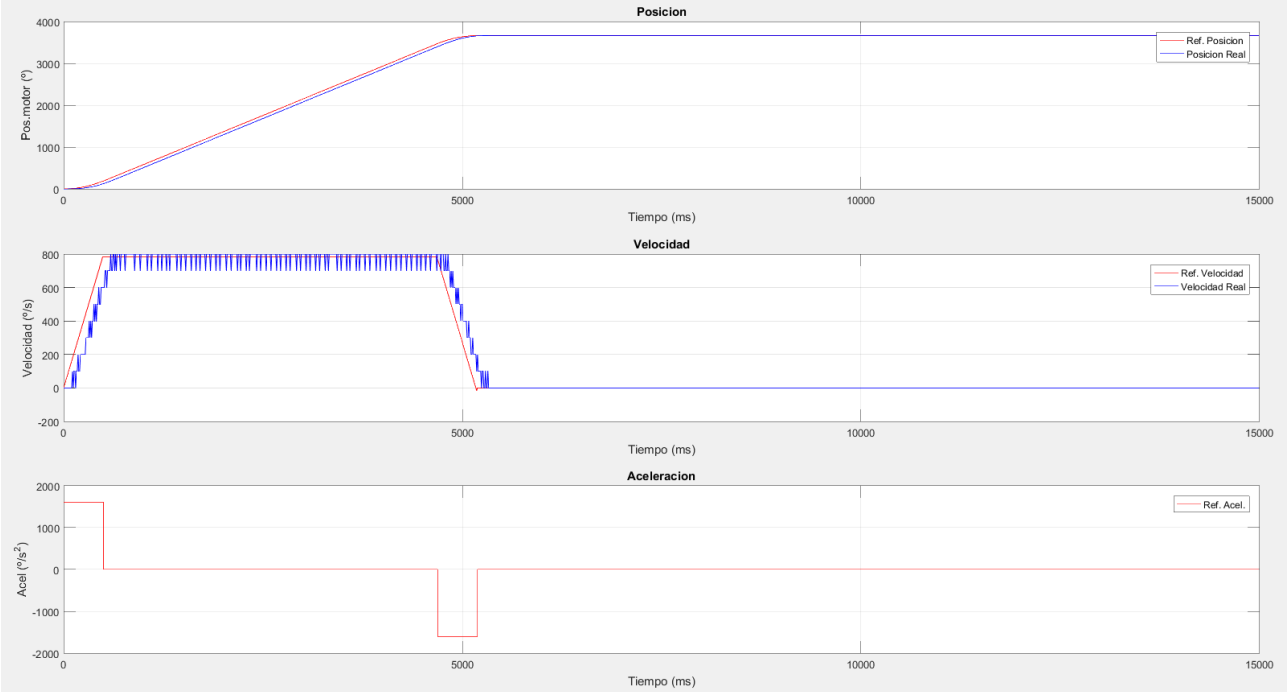


Figura 115: Resultado de implementación del control PI de velocidad con perfil de movimiento trapezoidal

Luego queda demostrada la posibilidad de control tanto en posición como en velocidad del sistema con los reguladores diseñados obteniendo resultados positivos y similares en ambos casos. La diferencia entre ambos queda reflejada en los índices de rendimiento, teniendo un menor error el control por velocidad para la referencia en rampa.

6. Conclusiones

6.1. Sobre el trabajo realizado

Este trabajo sienta las bases para el desarrollo en un microcontrolador ARM Cortex-M de un sistema de control en bucle cerrado para motores de corriente continua para el seguimiento de trayectorias punto a punto. Este supone solo un punto de partida para el desarrollo de un proyecto mucho más ambicioso que requiere de mayor cantidad de conocimientos, tiempo y recursos para su consecución.

El resultado del trabajo es evidente a la luz de los ensayos realizados. Las posibilidades de uso del mismo son tantas como se pueda querer desarrollar en cualquier ámbito. Desde la mejora y el desarrollo de máquinas de control numérico de tipo herramienta o de prototipado, como las actuales impresoras 3D que se encuentran limitadas a nivel comercial y de desarrollo por el sistema de movimiento a partir de bucle abierto de control de motores paso a paso, lo que limita la velocidad, la seguridad y la eficiencia de estas máquinas.

Otra posible aplicación es el ámbito educativo en carreras técnicas, como parte de prototipos de enseñanza de prácticas de laboratorio [JF 16].

Este trabajo permite comprobar, aplicar y complementar todos los conocimientos adquiridos en el curso del grado aunando en un mismo proyecto disciplinas como el control y automática, los accionamientos electromecánicos, la robótica, la electrónica y la informática industrial.

Si bien la gran cantidad de aspectos que pretende abarcar el proyecto impide el completo desarrollo de los mismos al nivel deseado, se puede concluir que este proyecto multidisciplinar alcanza sus objetivos y se consigue diseñar e implementar con resultados prácticos positivos un controlador programable en un microcontrolador de uso cada vez más extendido y con un nuevo sistema de *firmware* y librerías recién desarrollado y con pocas referencias, sirviendo este trabajo como referencia para futuros proyectos de similar naturaleza.

6.2. Futuras mejoras

Resultan evidentes, a la hora de concluir el trabajo la gran cantidad de variaciones y posibilidades que se podrían tener en cuenta y efectuar con el debido desarrollo. Sin embargo, para la naturaleza académica del trabajo y los medios disponibles para la realización del mismo, quedan fuera del lugar y pendientes, como en todo proyecto de desarrollo e implementación tras el estudio necesario y a su debido tiempo.

Entre las mejoras deseables conviene una mejor implementación en el microcontrolador del código. Pese a haber supuesto uno de los mayores desafíos del trabajo, y haber conseguido una buena implementación y ejecución, existen cantidad de aspectos a mejorar, empezando por la documentación y la elaboración de bibliotecas y funciones más sencillas de seguir y de administrar, así como la introducción de microkernels.

Conviene también por razones obvias, la implementación de una placa para la generación de ficheros en una unidad de memoria externa que facilite el uso y no consista en una solución temporal e improvisada.

La obtención del modelo del motor es la parte más importante para realizar el diseño del control óptimo. Pese a las distintas opciones implementadas y estudiadas, se sugiere no dejar de probar nuevos métodos en post del modelo más fiel a nuestro sistema real. Esto podría realizarse incluyendo las no linealidades como la zona muerta y la saturación en el modelo, pudiendo analizar el plano de fase (ω/e) y las funciones descriptivas de la saturación N(E,S).

La aplicación de la acción de control es, posiblemente, la causa de no haber conseguido el controlador más óptimo, si bien su desempeño queda patente. El integrado L298 presenta caídas de tensión en conmutación enormes (de hasta 0.7V) comparadas con otros drivers de puente-H completo como el 623 (de apenas un par de décimas de voltio). Si se pudieran dedicar los recursos necesarios a mejorar este bloque el control resultante mejoraría notablemente en tiempo de establecimiento y en reducción de errores que consideramos perturbaciones sin mayor análisis y se deben a fallos en la aplicación de la acción de control.

Se han indicado y planteado distintas soluciones a la problemática del diseño de un controlador, tanto analógico como discreto, a partir del modelo del motor de estudio. Análogamente, se han planteado distintas configuraciones de patrones de movimiento para la generación de referencias del control del proceso. Sin embargo solo se ha realizado la implementación de algunas de ellas para demostrar el resultado del trabajo realizado. El desarrollo e implementación de nuevas soluciones podría mejorar notablemente la calidad de la respuesta (con una aproximación de 4º orden en la generación de referencia de velocidad) y permitir reducir los parámetros de tiempo de establecimiento y sobre oscilación enormemente, o incluso evitar el efecto de perturbaciones (utilizando un regulador de mínima varianza con un modelo ARMAX).

7. Bibliografía

[Acc. Elec. 11] Manuel Sebastian Alvarez Alvarado. "Modelo matemático del motor CC separadamente excitado". Escuela Superior Politécnica del Litoral Ed. 2011.

[Acc. Elec. 15] Vicente Alfredo Aucejo. Accionamientos Electromecánicos. Máquinas de CC. Escuela Superior de Ingeniería del Diseño. UPV Ed. 2015.

[Anibal 1991]Aníbal Ollero Baturone. Control por computador: descripción interna y diseño óptimo. MARCOMBO Ed. 1991.

[CA 2014] Julián José Salt Llobregat, Ángel Cuenca Lacruz, Vicente Casanova Calvo, Antonio Correcher Salvador. Control Automático Tomos I y II (Tiempo Continuo y Tiempo Discreto). Ed. UPV, 2014.

[CA 2014] Julián José Salt Llobregat, y otros. Control Automático Tomos I y II (Tiempo Continuo y Tiempo Discreto). Ed. UPV, 2014.

[MAR 15] Marina Vallés Miquel. Control avanzado por computador. Escuela Técnica Superior de Ingeniería del Diseño. UPV Ed. 2015.

[RAU 14] Raúl Simarro Fernández. Control de sistemas mecatrónicos (cód. 12175). Escuela Técnica Superior de Ingeniería del Diseño. UPV Ed. 2014.

[Moog 16] Moog Animatic Catalog. Avaliable through http://onexia.com/ at (01/06/16)

[Ogata 1996] Ogata, K. "Sistemas de control en tiempo discreto". Ed. Pearson Education, 1996.

[Reprap16] http://reprap.org/wiki/Triffid_Hunter's_Calibration_Guide/es. (Consultada 18/05/16)

[SR 15] Zotovic Stanisic, Ranko y otros. Sistemas robotizados. Escuela Técnica Superior de Ingeniería del Diseño. UPV Ed. 2015

[Serrano 1989]Luis Serrano Iribarnegaray. "Fundamentos de máquinas eléctricas rotativas"Ed. MARCOMBO, S.A. 1989

[STM 15] RM009. STM32405_407_427_429_Reference_manual Edición de octubre de 2015. Disponible en: STMicroelectronics web site (*http://www.st.com*).

[STM 16] STM32F29xx. Datasheet. Edición de enero de 2016. Disponible en: STMicroelectronics web site (*http://www.st.com*).

[TAL 76] Tal, J.; (1976). Design and Analysis of Pulsewidth-Modulated Amplifiers for DC Servo Systems, IEEE Trans. on Industrial Electronics and Control Instrumentation, Vol IECI 23, No.1

[JAHNS 84] Jahns, T.M.; (1984). Torque Production in Permanent Magnet Synchronous Motor Drives with Rectangular Current Excitation, IEEE Trans. on Industry Applications, Vol IA-20, No. 4.

[KIM 08] Kim Doang Nguyen; Teck-Chew Ng and I-Ming; "On Algorithms for Planning S-

curve Motion Profiles" International Journal of Advanced Robotic Systems, Vol. 5, No. 1 (2008)

[CHUCK 07] Chuck Lewin *"Mastering motion profiles"* Performance Motion Devices Inc (PMD). Machine design. Febrero de 2007.

[JF 16] González Rojo, J.F.; Sánchez Salmerón, A.J. Prototipo de visión artificial de bajo coste para desarrollo de prácticas docentes. Jornadas de automática XXXVII

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR PROGRAMABLE DE PATRONES DE MOVIMIENTO PARA MOTORES DE CORRIENTE CONTINUA CON APLICACIÓN EN ROBOTS Y MÁQUINAS CNC

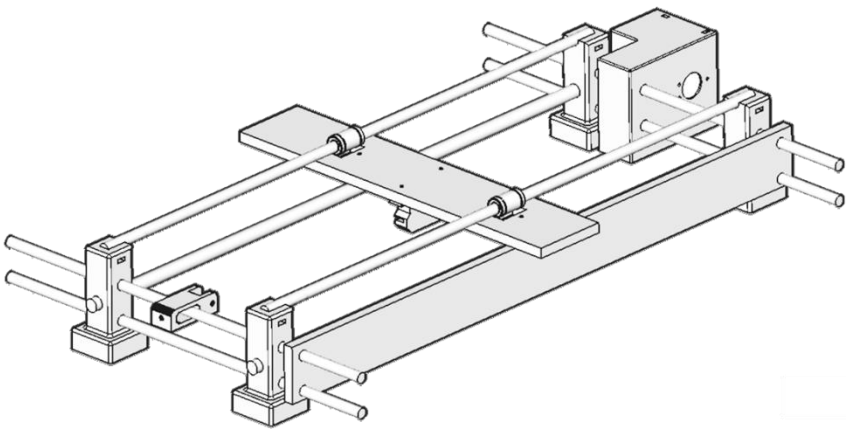
2.PLANOS

Autor:
D. José Félix González Rojo

Tutor:
D. Ángel Perles Ivars

Cotutor:
D. Miguel Sánchez López

Valencia, julio de 2016



Índice de planos

1. Plano eléctrico

1.1. Diagrama eléctrico. Conexiones ..... Plano 1

2. Plano mecánico

2.1. Vista general ..... Plano 2

2.2. Eje ..... Plano 3

2.2.1. Estructura ..... Plano 4

2.2.1.1. Esquinas ..... Plano 5

2.2.1.1. Varillas ..... Plano 6

2.2.2. Soporte motor ..... Plano 7

2.2.3. Soporte tensor ..... Plano 8

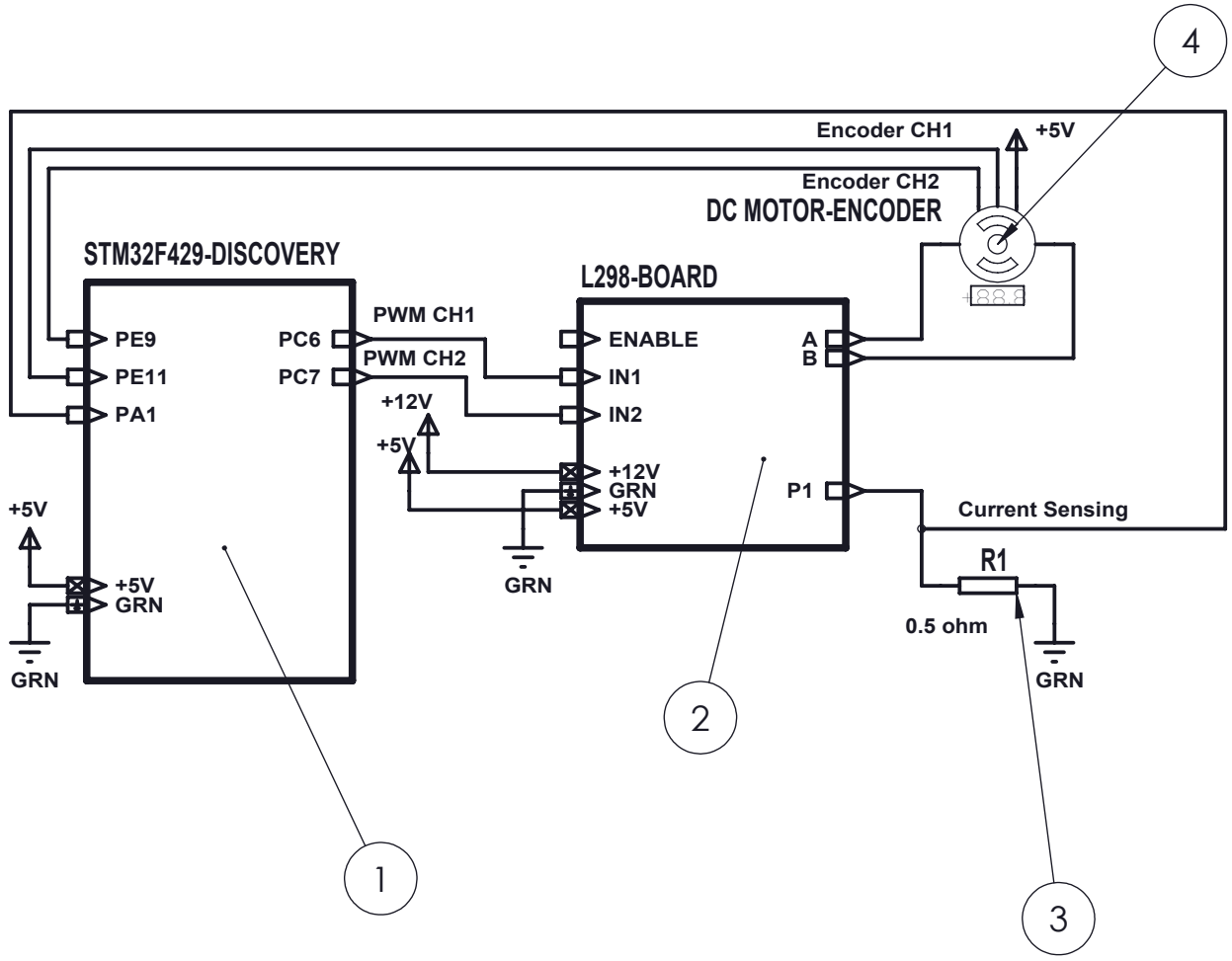
2.2.4. Lateral ..... Plano 9

2.2.5. Rodamientos ..... Plano 10

2.3. Carro ..... Plano 11

2.3.1. Base ..... Plano 12

2.3.2. Soporte correa ..... Plano 13



ELEMENTO	DESCRIPCIÓN	PLANO	CANTIDAD
4	DC Brushed Motor and Encod.	-	1
3	Current Sensing Resistor	-	1
2	L298-BOARD	-	1
1	STM32F4-DISCOVERY	-	1

TÍTULO PROYECTO: DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR PROGRAMABLE DE PATRONES DE MOVIMIENTO PARA MOTORES DE CORRIENTE CONTINUA CON APLICACIÓN EN ROBOTS Y MÁQUINAS CNC

FECHA: 10/06/2016

FECHA REV: 20/06/20165

REFERENCIA: 16-01

TAMAÑO: A4

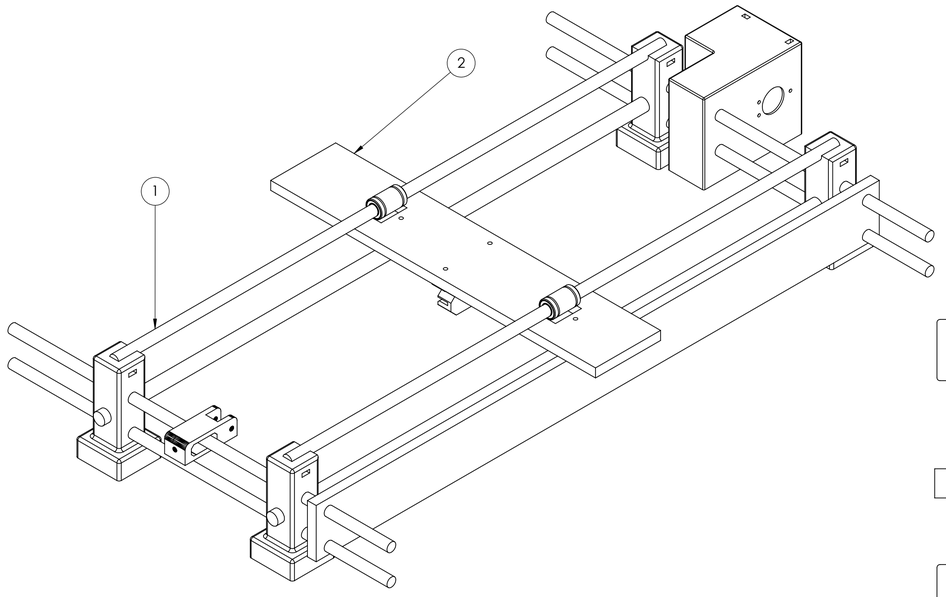
AUTOR: Jose Felix Garcia Rojo

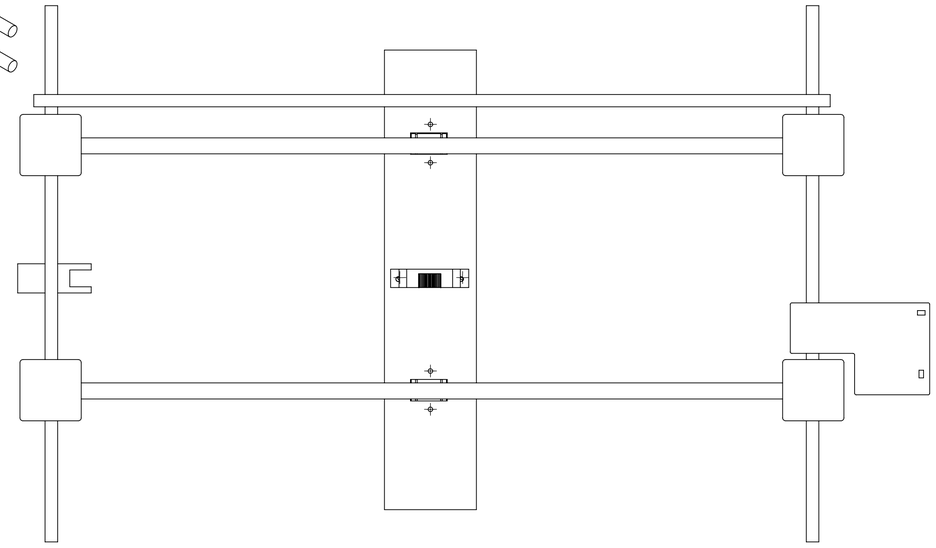
FIRMA Y SELLO:

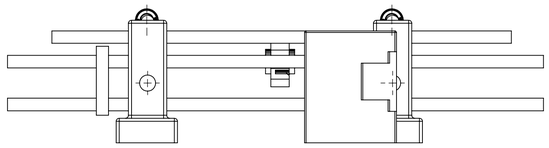
ESCALA: 1:1

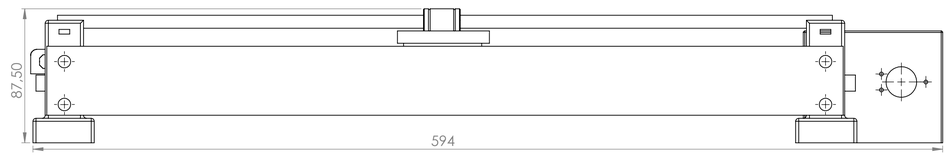
NOMBRE PLANO: Diagrama Electrico. Conexiones

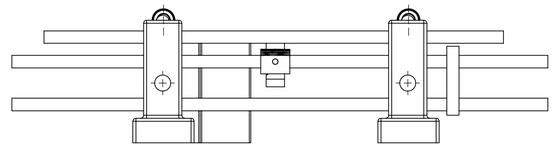
Nº PLANO: 1

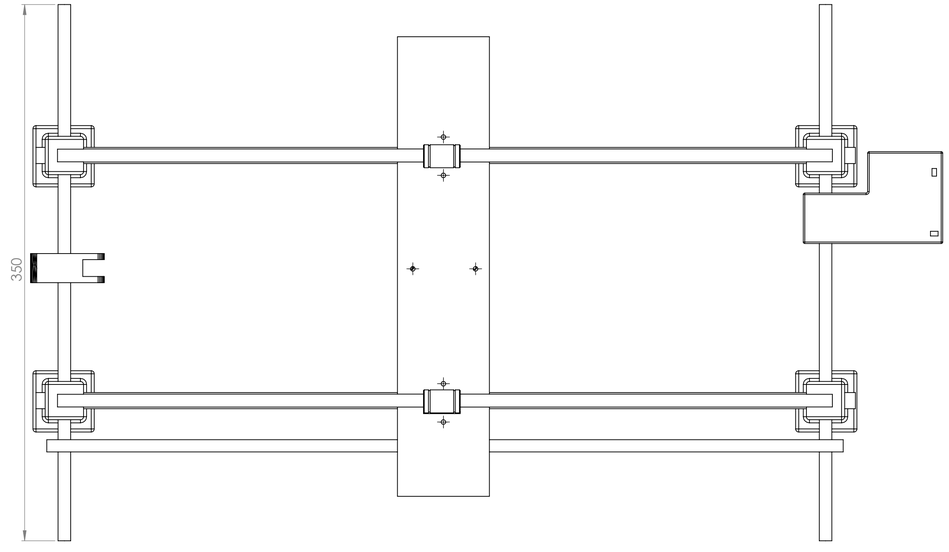






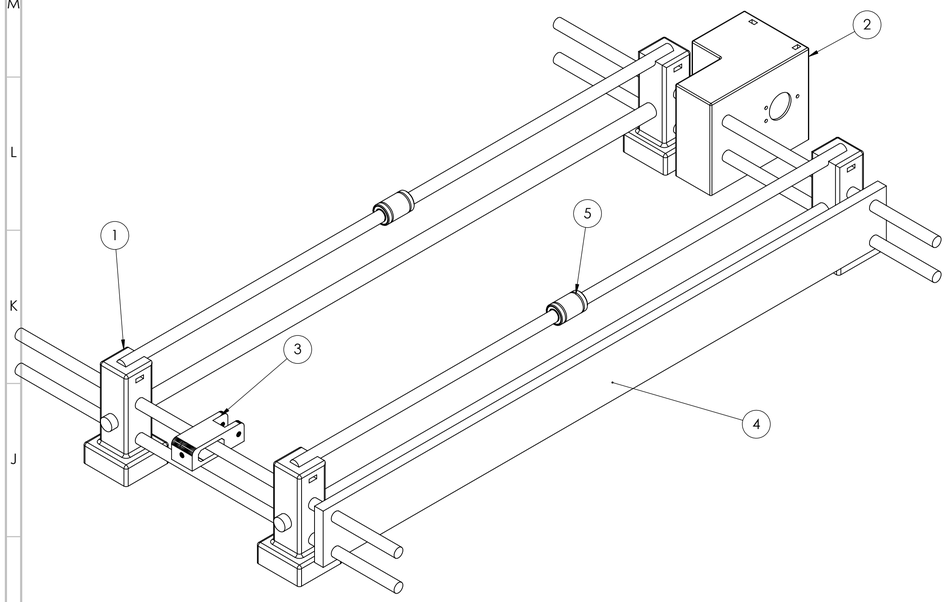


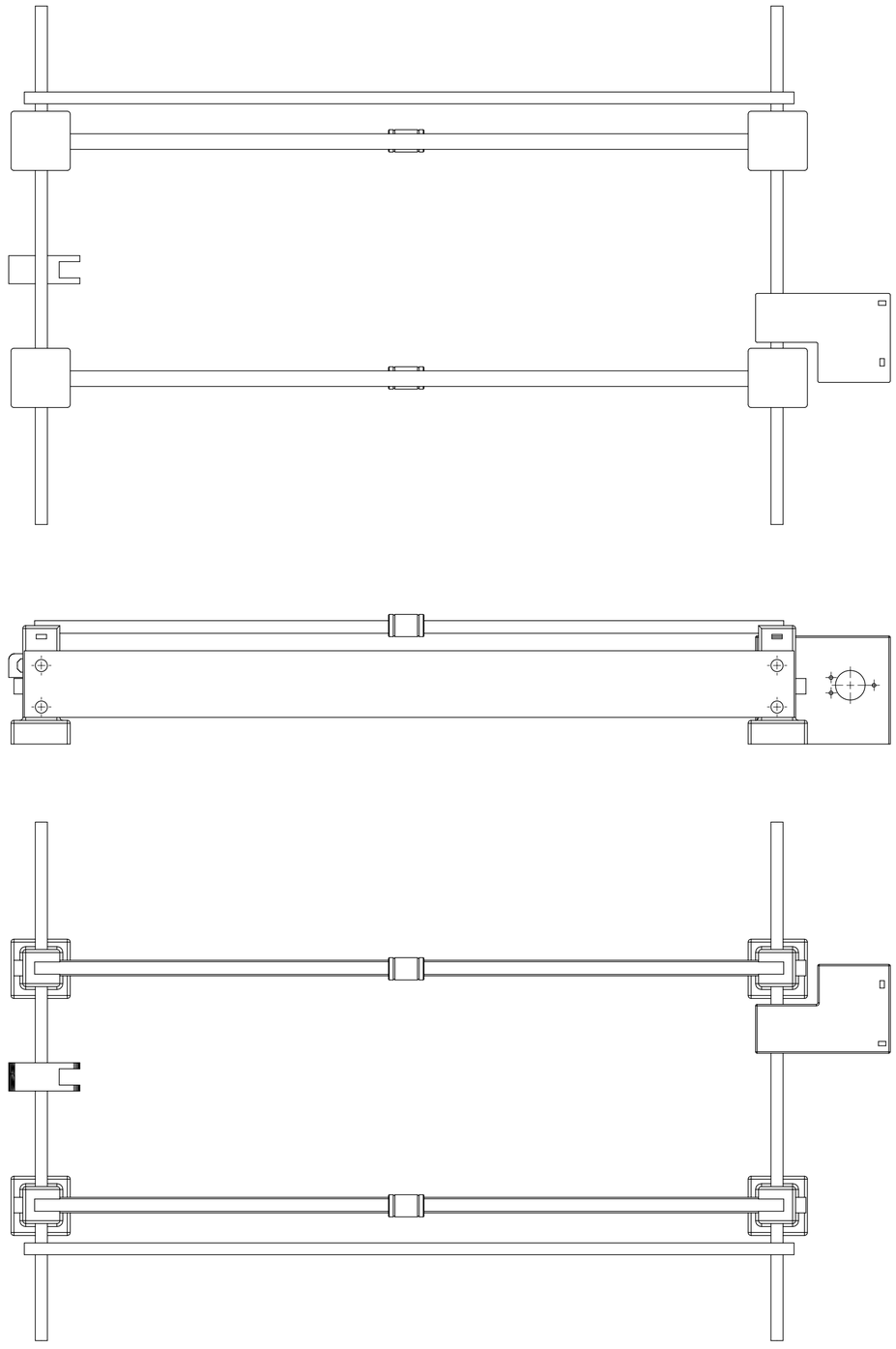


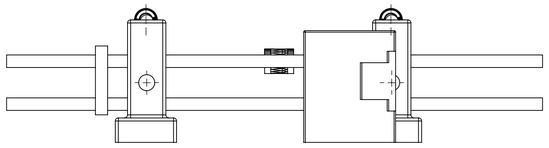


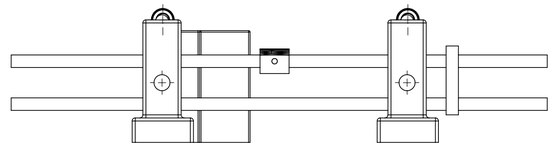
2	Carro	11-13C7	1
1	Eje	3-10	1
ELEMENTO	DESCRIPCIÓN	PLANO	CANTIDAD

TÍTULO PROYECTO: DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR PROGRAMABLE DE PATRONES DE MOVIMIENTO PARA MOTORES DE CORRIENTE CONTINUA CON APLICACIÓN EN ROBOTS Y MÁQUINAS CNC		FECHA: 10/06/2016
		FECHA REV: 20/06/20165
REFERENCIA: 16-002	TAMAÑO: A1	ESCALA: 1:2
AUTOR: Jose Felix Gonzalez Rojo	FIRMA Y SELLO:	
NOMBRE PLANO: Vista General		Nº PLANO: 2



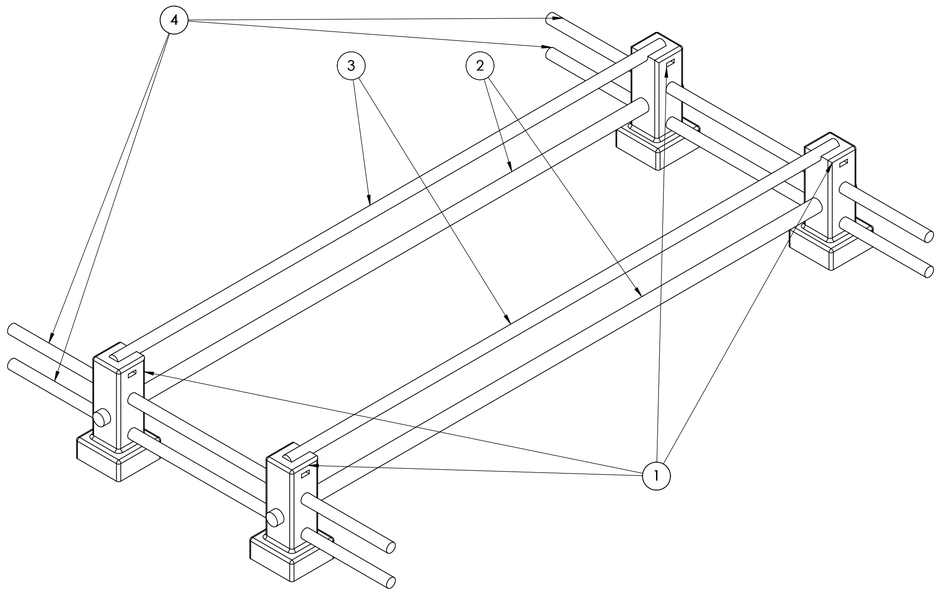


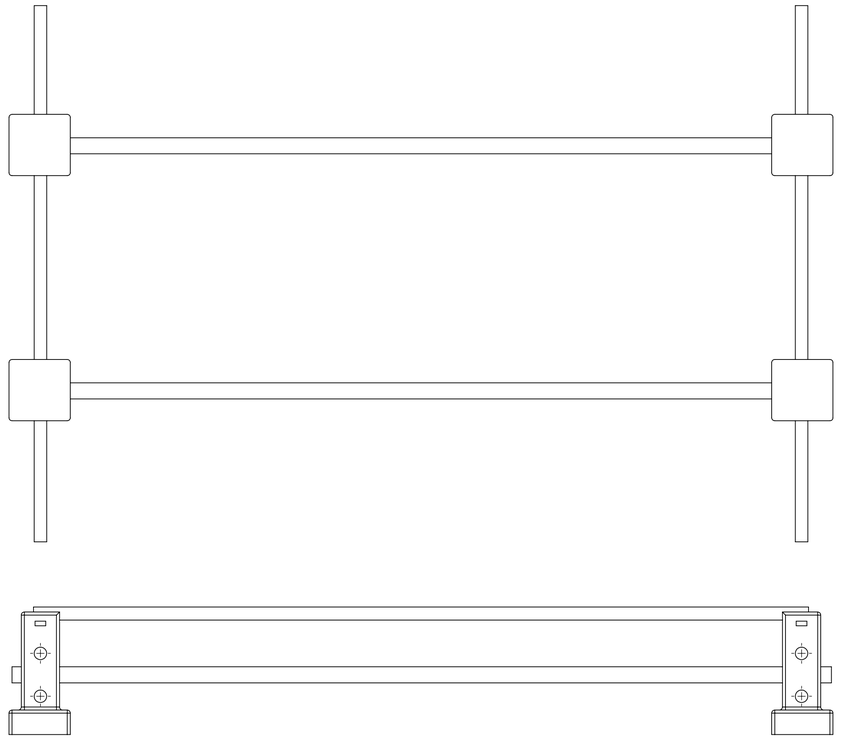


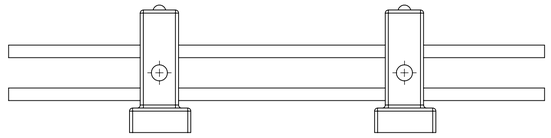


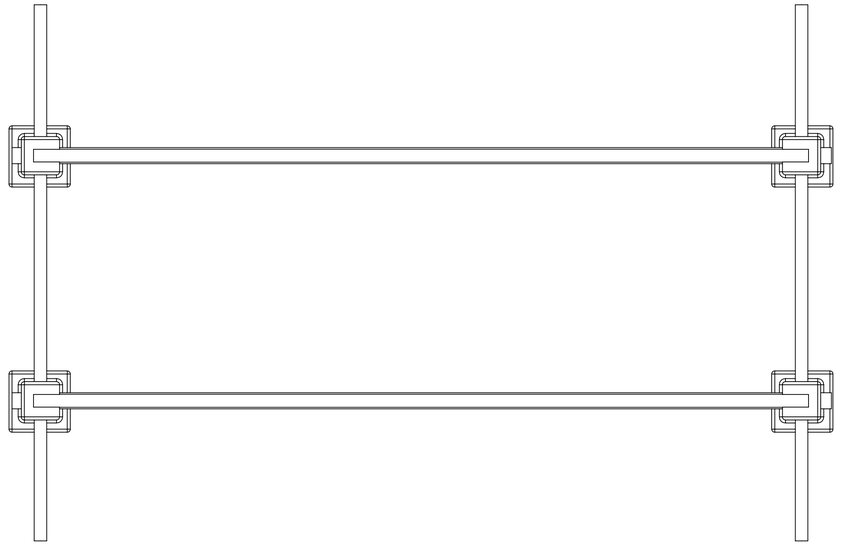
5	Rodamientos	10	
4	Lateral	9	1
3	Soporte Tensor	8	1
2	Soporte Motor	7	1
1	Estructura	4-6	1
ELEMENTO	DESCRIPCIÓN	PLANO	CANTIDAD

TÍTULO PROYECTO: DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR PROGRAMABLE DE PATRONES DE MOVIMIENTO PARA MOTORES DE CORRIENTE CONTINUA CON APLICACIÓN EN ROBOTS Y MÁQUINAS CNC		FECHA: 10/06/2016
		FECHA REV: 20/06/20165
REFERENCIA: 16-003	TAMAÑO: A1	ESCALA: 1:2
AUTOR: Jose Felix Gonzalez Rojo	FIRMA Y SELLO:	
NOMBRE PLANO: Eje		Nº PLANO: 3



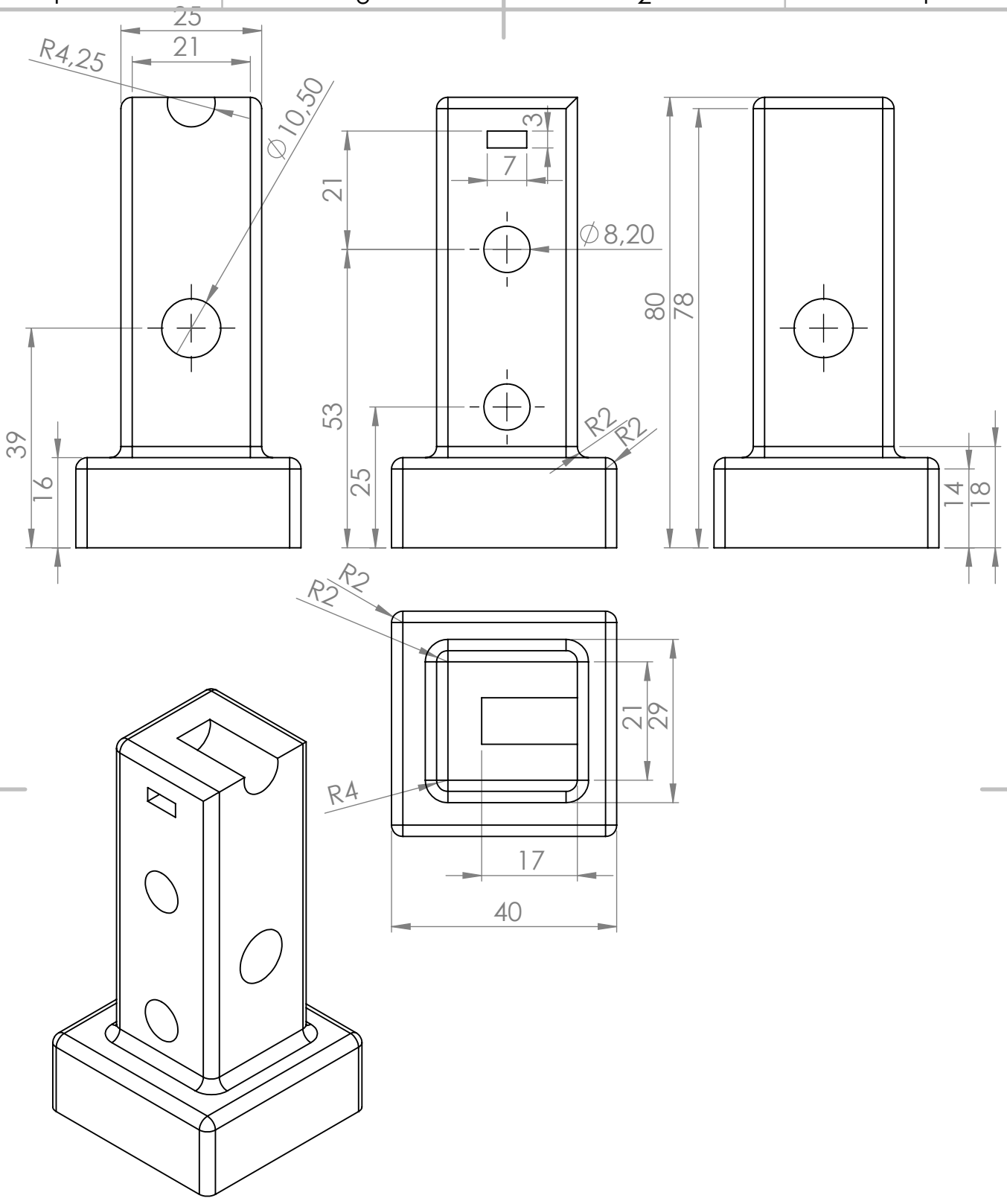




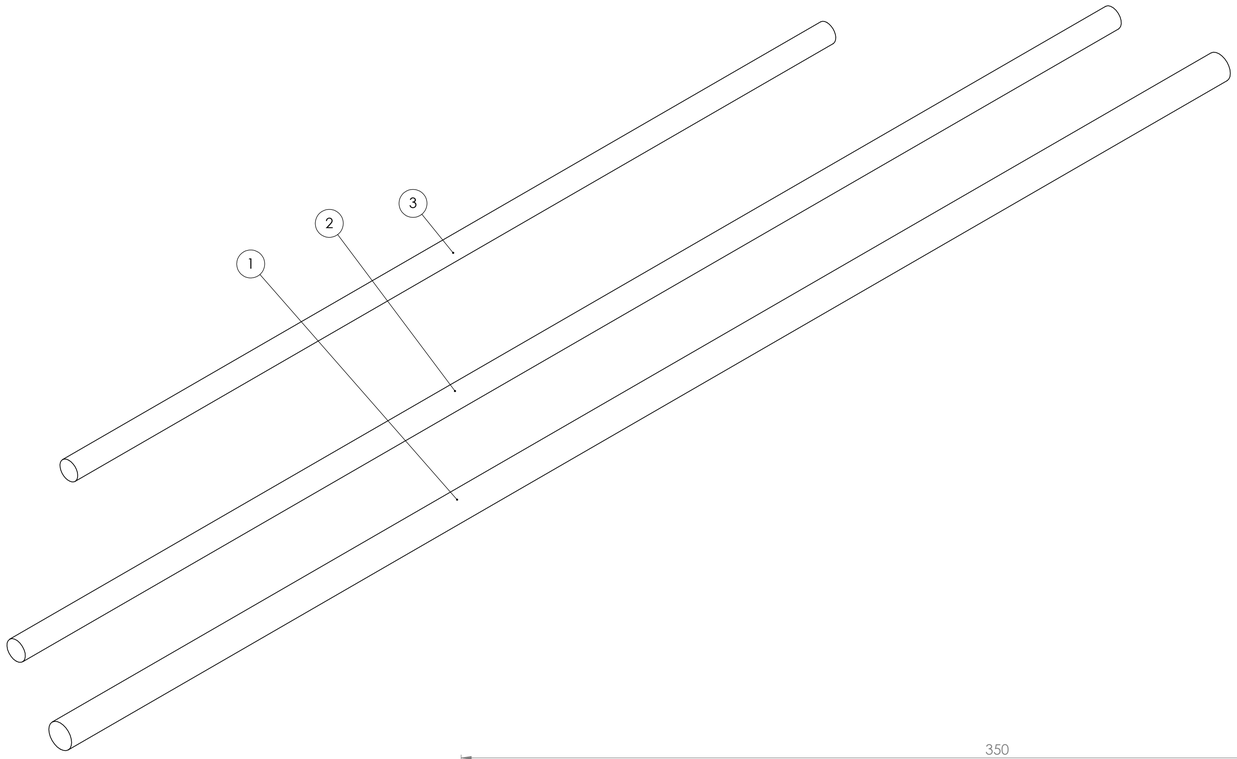


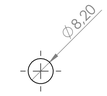
4	Varilla roscada M8	6	4
3	Varilla lisa M8	6	2
2	Varilla roscada M10	6	2
1	Esquinas	5	4
ELEMENTO	DESCRIPCIÓN	PLANO	CANTIDAD

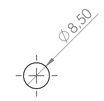
TÍTULO PROYECTO: DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR PROGRAMABLE DE PATRONES DE MOVIMIENTO PARA MOTORES DE CORRIENTE CONTINUA CON APLICACIÓN EN ROBOTS Y MÁQUINAS CNC	FECHA: 10/06/2016
	FECHA REV: 20/06/20165
REFERENCIA: 16-004 / TAMAÑO: A1	ESCALA: 1:1
AUTOR: Jose Felix Gonzalez Rojo / FIRMA Y SELLO:	Nº PLANO: 4
NOMBRE PLANO: Estructura	

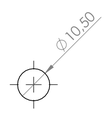


TÍTULO PROYECTO: DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR PROGRAMABLE DE PATRONES DE MOVIMIENTO PARA MOTORES DE CORRIENTE CONTINUA CON APLICACIÓN EN ROBOTS Y MÁQUINAS CNC	FECHA: 10/06/2016
	FECHA REV: 20/06/20165
REFERENCIA: 16-005 / TAMAÑO: A4	ESCALA: 1:1
AUTOR: Jose Felix González Rojo / FIRMA Y SELLO:	
NOMBRE PLANO: Esquina	Nº PLANO: 5



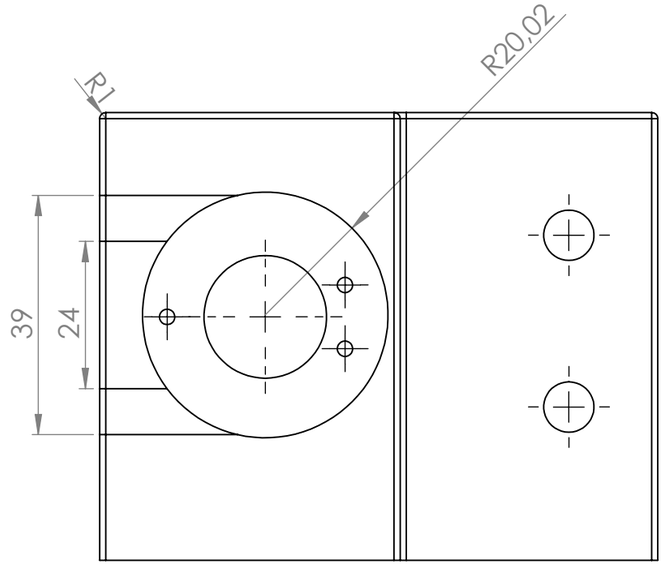


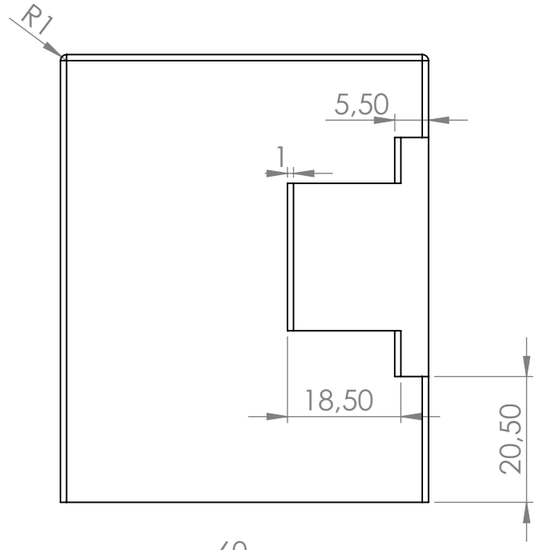


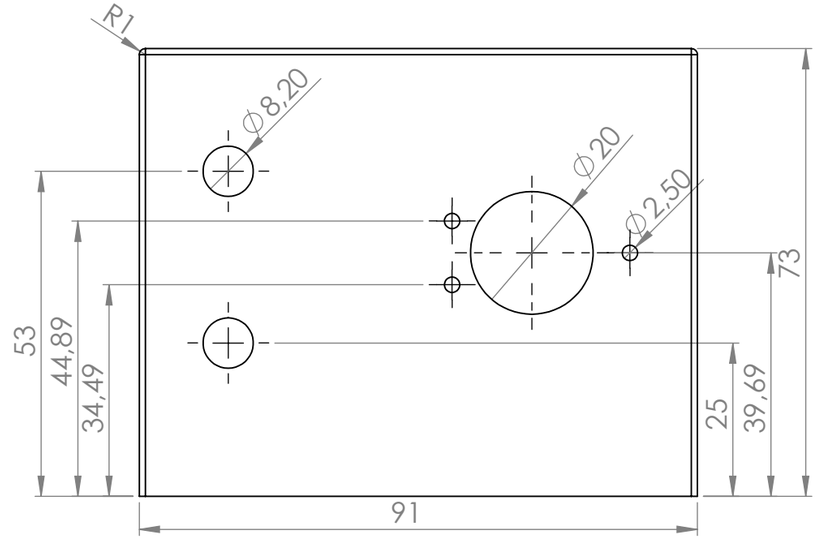


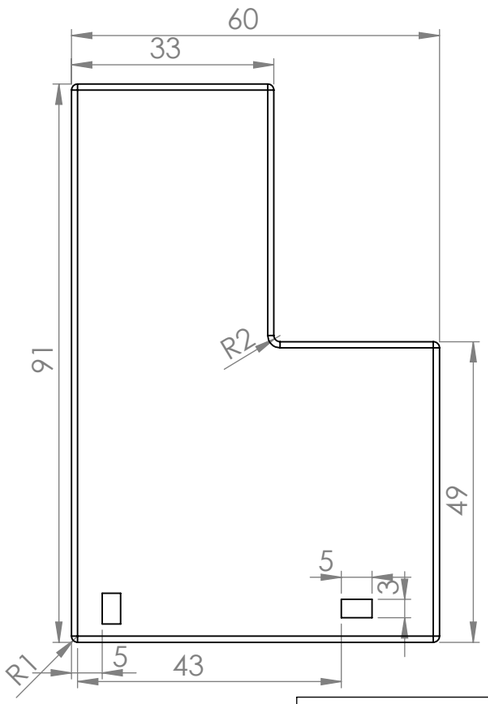
3	Varilla roscada M8	6	4
2	Varilla lisa M8	6	2
1	Varilla roscada M10	6	2
ELEMENTO	DESCRIPCIÓN	PLANO	CANTIDAD

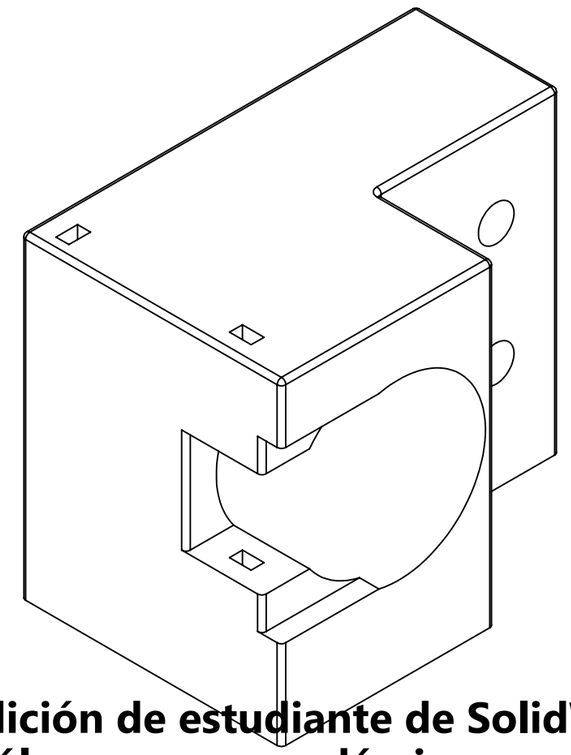
TÍTULO PROYECTO: DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR PROGRAMABLE DE PATRONES DE MOVIMIENTO PARA MOTORES DE CORRIENTE CONTINUA CON APLICACIÓN EN ROBOTS Y MAQUINAS CNC		FECHA: 10/06/2016
		FECHA REV: 20/06/20165
REFERENCIA: 16-006	TAMAÑO: A1	ESCALA: 1:1
AUTOR: Jose Felix Gonzalez Rojo	FIRMA Y SELLO:	
NOMBRE PLANO: Varillas		Nº PLANO: 6





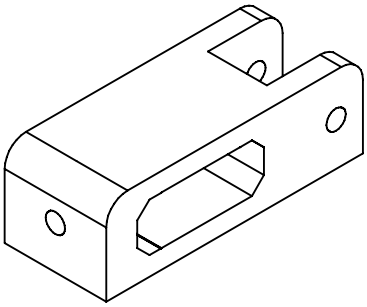


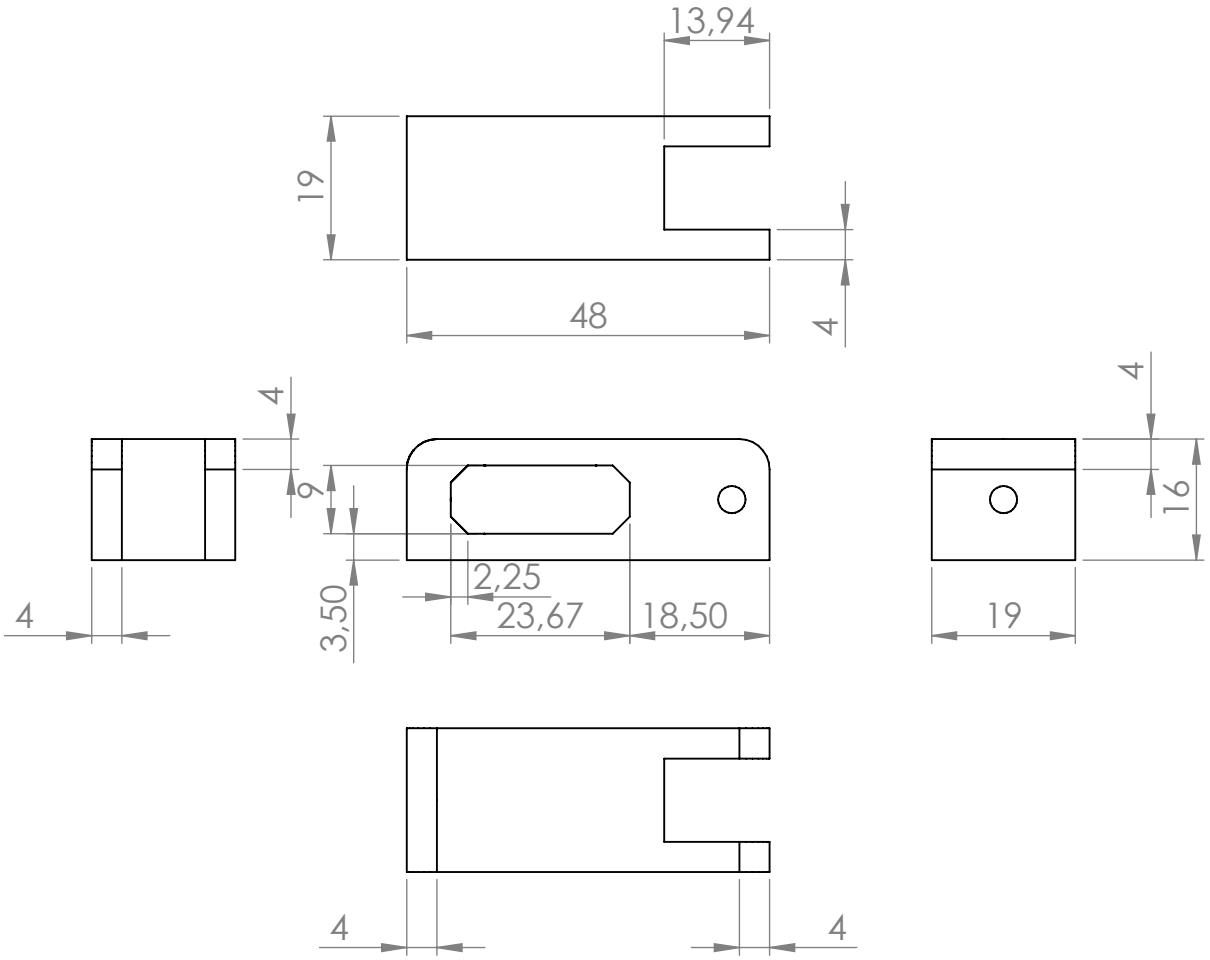




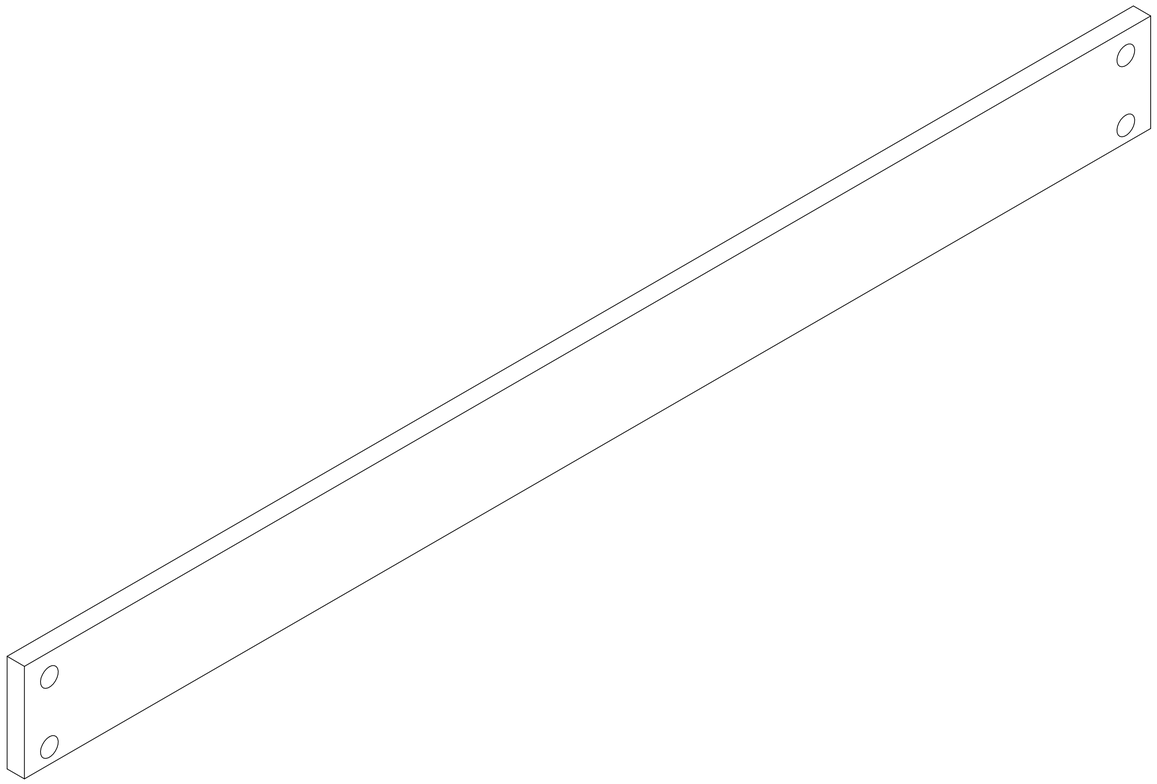
Edición de estudiante de SolidWorks.
Sólo para uso académico.

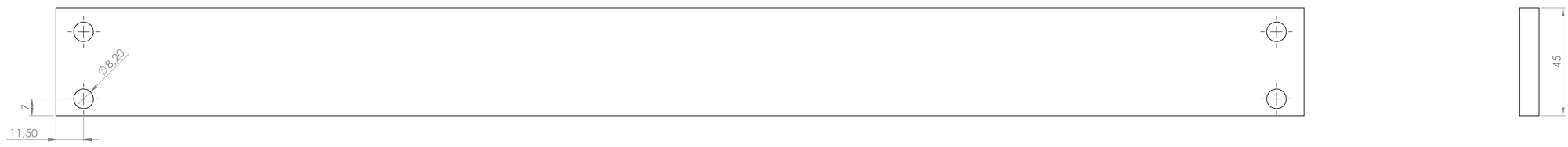
TÍTULO PROYECTO: DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR PROGRAMABLE DE PATRONES DE MOVIMIENTO PARA MOTORES DE CORRIENTE CONTINUA CON APLICACIÓN EN ROBOTS Y MÁQUINAS CNC		FECHA: 10/06/2016
		FECHA REV: 20/06/2016
REFERENCIA: 16-007	TAMAÑO: A3	ESCALA: 1:1
AUTOR: Jose Felix Gonzalez Rojo	FIRMA Y SELLO:	
NOMBRE PLANO: Soporte Motor		Nº PLANO: 7





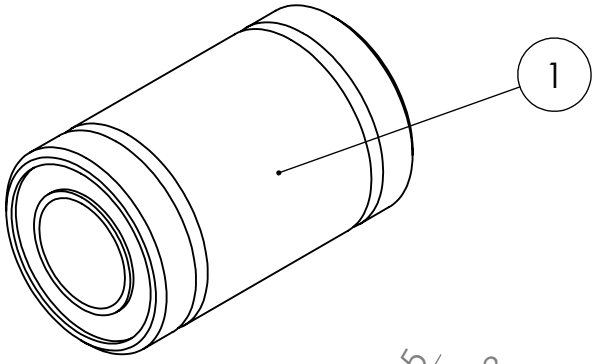
TÍTULO PROYECTO: DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR PROGRAMABLE DE PATRONES DE MOVIMIENTO PARA MOTORES DE CORRIENTE CONTINUA CON APLICACIÓN EN ROBOTS Y MÁQUINAS CNC	FECHA: 10/06/2016
	FECHA REV: 20/06/20165
REFERENCIA: 16-008 / TAMAÑO: A4	ESCALA: 1:1
AUTOR: Jose Felix González Rojo / FIRMA Y SELLO:	
NOMBRE PLANO: Tensor Correa	Nº PLANO: 8

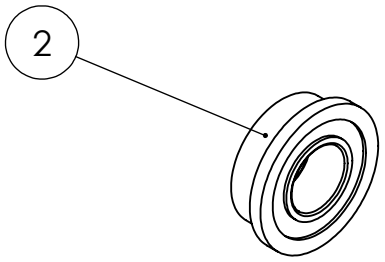


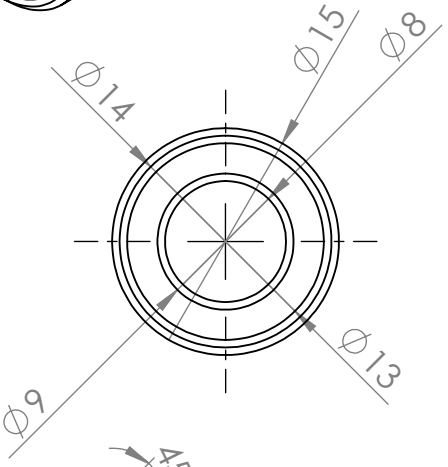


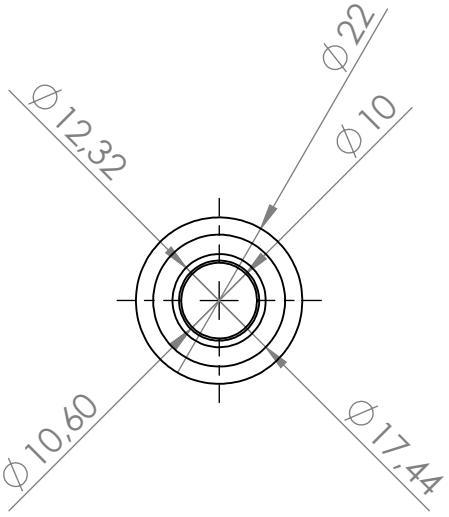
Edición de estudiante de SolidWorks.
Sólo para uso académico.

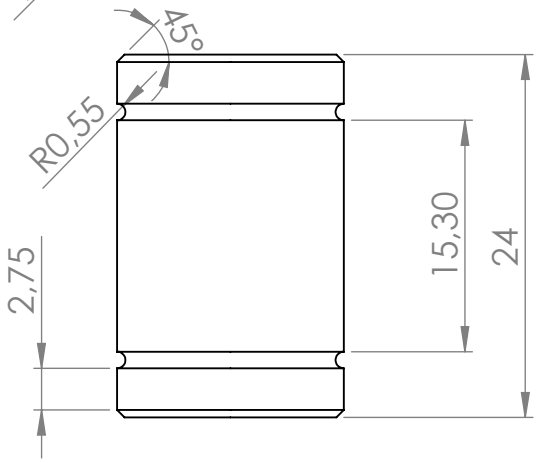
TÍTULO PROYECTO: DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR PROGRAMABLE DE PATRONES DE MOVIMIENTO PARA MOTORES DE CORRIENTE CONTINUA CON APLICACIÓN EN ROBOTS Y MÁQUINAS CNC	FECHA: 10/06/2016
	FECHA REV: 20/06/20165
REFERENCIA: 16-009 — TAMAÑO: A1	ESCALA: 1:1
AUTOR: Jose Felix Gonzalez Rojo — FIRMA Y SELLO:	Nº PLANO: 9
NOMBRE PLANO: Lateral	

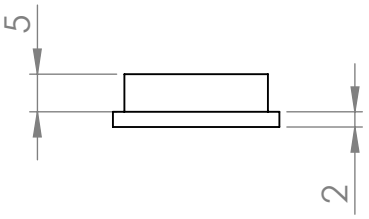






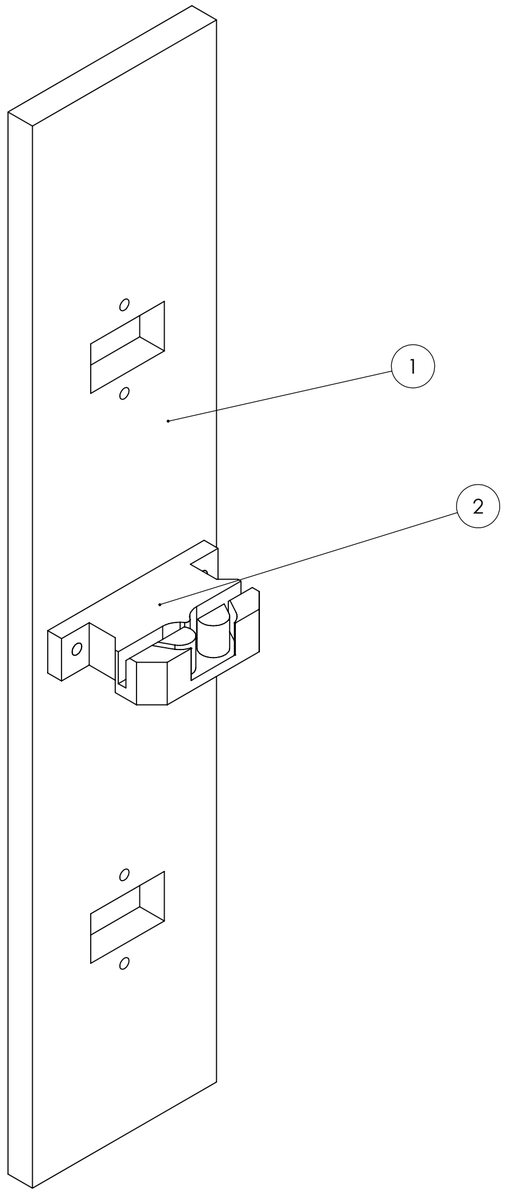


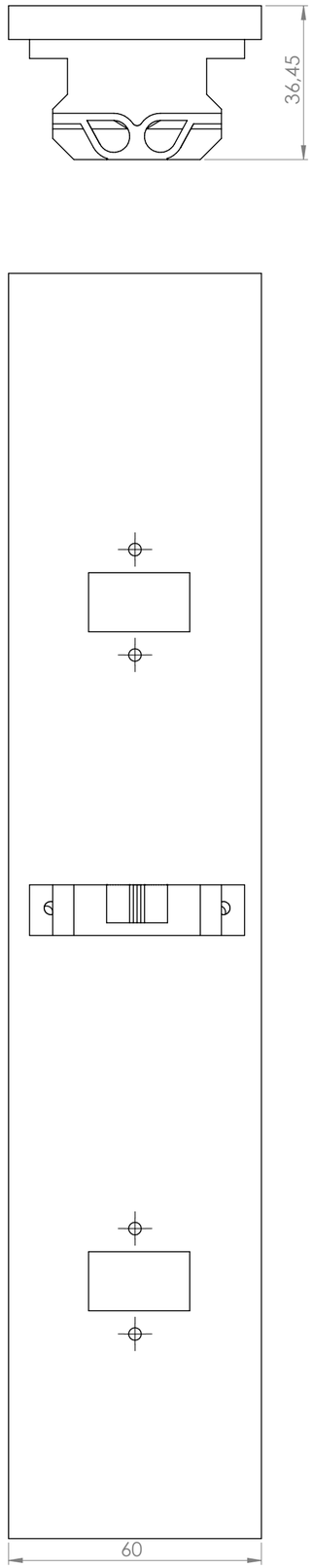


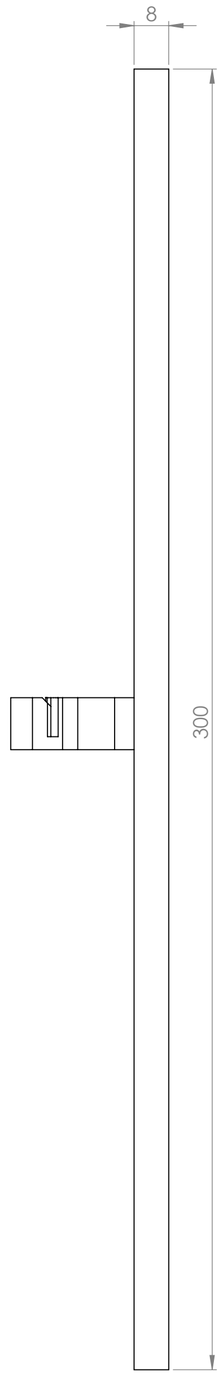


ELEMENTO	DESCRIPCIÓN	PLANO	CANTIDAD
2	F623ZZ	10	2
1	LM8UU	10	2

TÍTULO PROYECTO: DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR PROGRAMABLE DE PATRONES DE MOVIMIENTO PARA MOTORES DE CORRIENTE CONTINUA CON APLICACIÓN EN ROBOTS Y MÁQUINAS CNC	FECHA: 10/06/2016
	FECHA REV: 20/06/20165
REFERENCIA: 16-010 / TAMAÑO: A4	ESCALA: 2:1
AUTOR: Jose Felix González Rojo / FIRMA Y SELLO:	
NOMBRE PLANO: Rodamientos	Nº PLANO: 10

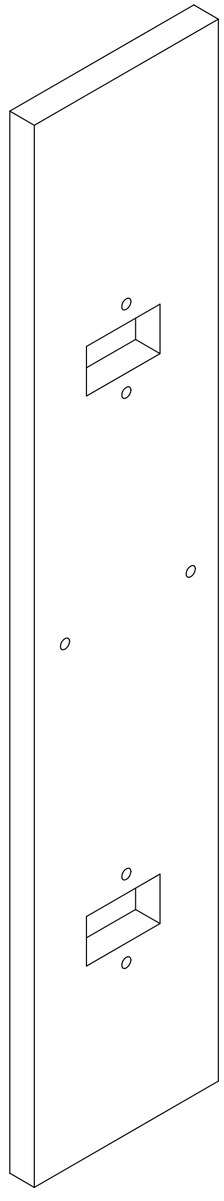


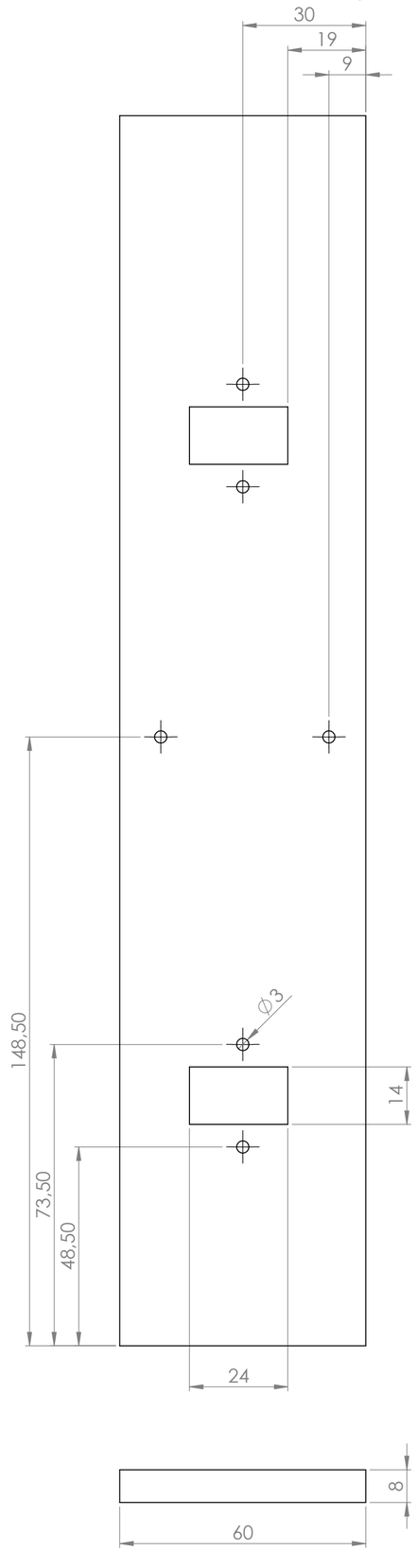


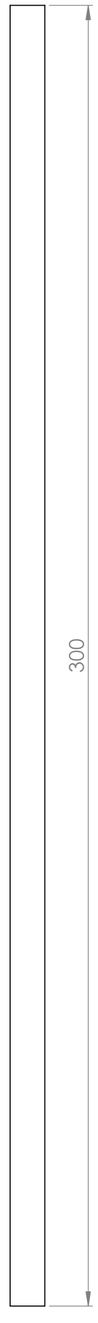


2	Soporte Correa	11	1
1	Base	11	1
ELEMENTO	DESCRIPCIÓN	PLANO	CANTIDAD

TÍTULO PROYECTO: DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR PROGRAMABLE DE PATRONES DE MOVIMIENTO PARA MOTORES DE CORRIENTE CONTINUA CON APLICACIÓN EN ROBOTS Y MÁQUINAS CNC	FECHA: 10/06/2016
	FECHA REV: 20/06/20165
REFERENCIA: 16-011 / TAMAÑO: A2	ESCALA: 1:1
AUTOR: Jose Felix Gonzalez Rojo / FIRMA Y SELLO:	
NOMBRE PLANO: Carro	Nº PLANO: 11

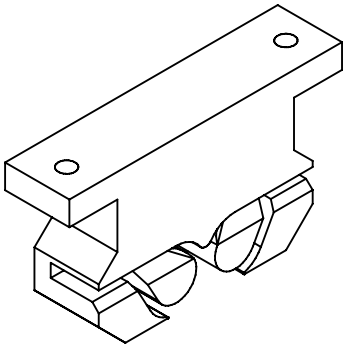


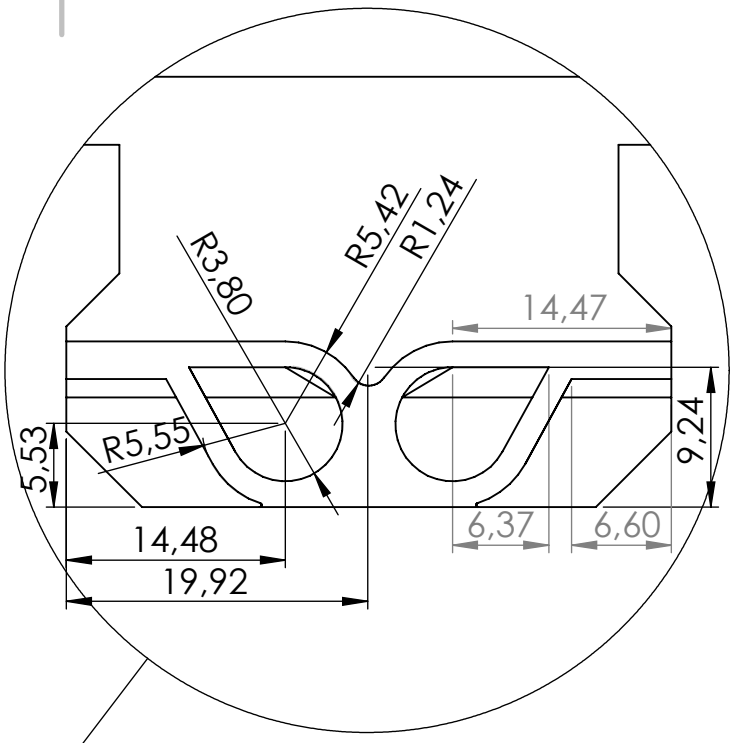




Edición de estudiante de SolidWorks.
Sólo para uso académico.

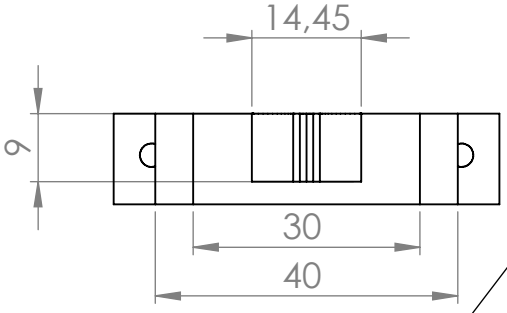
TÍTULO PROYECTO: DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR PROGRAMABLE DE PATRONES DE MOVIMIENTO PARA MOTORES DE CORRIENTE CONTINUA CON APLICACIÓN EN ROBOTS Y MÁQUINAS CNC		FECHA: 10/06/2016
		FECHA REV: 20/06/20165
REFERENCIA: 16-012	TAMAÑO: A2	ESCALA: 1:1
AUTOR: Jose Felix Gonzalez Rojo	FIRMA Y SELLO:	
NOMBRE PLANO: Base		Nº PLANO: 12

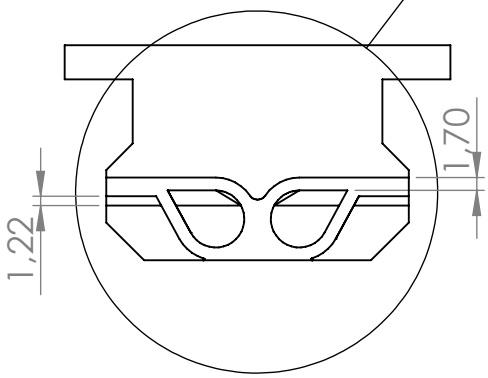


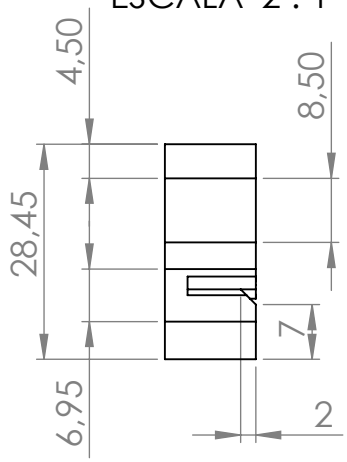


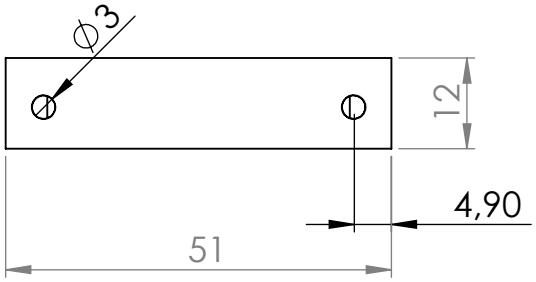
DETALLE A

ESCALA 2 : 1









TÍTULO PROYECTO: DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR PROGRAMABLE DE PATRONES DE MOVIMIENTO PARA MOTORES DE CORRIENTE CONTINUA CON APLICACIÓN EN ROBOTS Y MÁQUINAS CNC	FECHA: 10/06/2016
	FECHA REV: 20/06/20165
REFERENCIA: 16-013 / TAMAÑO: A4	ESCALA: 1:1
AUTOR: Jose Felix González Rojo / FIRMA Y SELLO: **Edición de estudiante de SolidWorks. Sólo para uso académico.**	Nº PLANO: 13
NOMBRE PLANO: Soporte Correa	

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR PROGRAMABLE DE PATRONES DE MOVIMIENTO PARA MOTORES DE CORRIENTE CONTINUA CON APLICACIÓN EN ROBOTS Y MÁQUINAS CNC

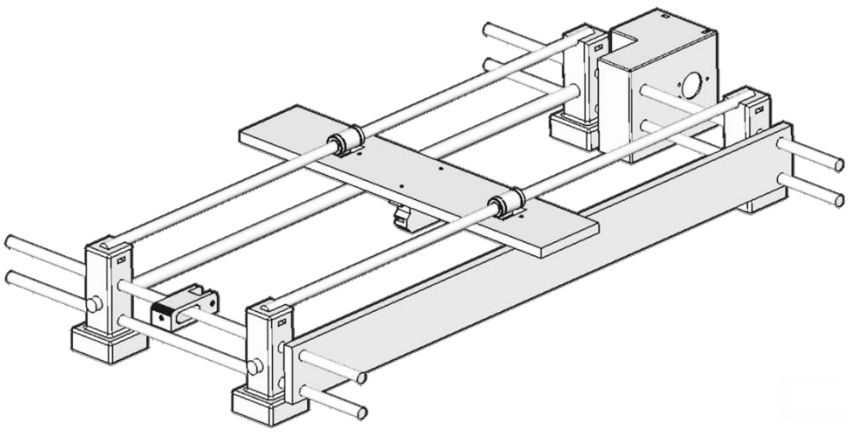
3.PLIEGO DE CONDICIONES

Autor:
D. José Félix González Rojo

Tutor:
D. Ángel Perles Ivars

Cotutor:
D. Miguel Sánchez López

Valencia, julio de 2016



Contenido del pliego de condiciones

1. Definición y alcance del pliego 2

2. Condiciones y normas de carácter general 3

1.1. Instalación 3

1.2. Seguridad 3

1.3. Utilización 4

1.4. Mantenimiento 4

3. Condiciones particulares 5

3.1. Técnicas 5

3.1.1. Microcontrolador 5

3.1.2. Etapa de potencia 7

3.1.3. Motor, bloque reductor y *encoder* 7

3.1.4. Estructura 8

3.2. Legales 9

1. Definición y alcance del pliego

El presente proyecto supone el diseño e implementación para testeo del mismo de un controlador programable de patrones de movimiento para un motor de corriente continua que puede ser el actuador responsable del movimiento de cualquier configuración de robot o máquina de control numérico por computador presente en la industria. Si bien el planteamiento del trabajo es académico, tiene en cuenta las necesidades reales actuales del campo de la robótica industrial aplicada a elementos como puede ser el de las máquinas herramienta de producción mediante mecanizado o de prototipado mediante impresión 3D.

Este controlador se programa para la arquitectura ARM basándose en el uso de las bibliotecas HAL (Hardware Abstractión Layer) y el estándar CMSIS. No obstante, la aplicación a microcontroladores de arquitectura y prestaciones similares es sencilla, suponiendo el presente proyecto una biblioteca para la sensorización y control completos de un motor de corriente continua, siendo solo necesario realizar la variación en la inicialización y uso de los periféricos. Análogamente, la adaptación a nuevas disposiciones de etapas de potencia o acondicionamiento de la señal de control solo requieren de la conversión a la hora de transmitir la solución del control al periférico en cuestión.

Para el desarrollo del software se tendrá en consideración las características del prototipo de pruebas, incluyendo motor (DC040B Series Pittman), encoder (E30 Pittman) y reductora(G30A), así como el microcontrolador (STM32F429-Discovery) y la etapa de potencia (L298).

Pese a las citadas posibilidades de compatibilidad el correcto funcionamiento de este proyecto solo se ha comprobado con la configuración de elementos que se ha indicado y cuyas características técnicas se detallan a continuación.

El objeto de este documento, por tanto, es fijar los criterios técnicos a tener en cuenta y que sean de utilidad para la reproducción, desarrollo o implementación del proyecto: “Diseño e implementación de un controlador programable de patrones de movimiento para motores de corriente continua con aplicación en robots y máquinas CNC”, de acuerdo con las especificaciones recogidas en el presente pliego técnico. Estas supondrán valores comprobados de durabilidad, fiabilidad y seguridad.

El ámbito de aplicación de este documento se extiende a todos los sistemas mecánicos, eléctricos y electrónicos que forman parte del proyecto. En determinados supuestos se podrán adoptar, por la propia naturaleza del mismo o del desarrollo tecnológico, soluciones diferentes a las propuestas, siempre que quede suficientemente justificada su necesidad y que no impliquen una disminución de las exigencias mínimas de calidad esperada en el mismo.

2. Condiciones y normas de carácter general

A continuación se exponen una serie de condiciones y normas de carácter general.

1.1. Instalación

Se debe comprobar si se cumplen los requisitos para la instalación en cuanto a hardware y software.

Se debe realizar la implementación o desarrollo de este trabajo siguiendo las indicaciones del mismo.

Para el seguimiento y desarrollo del proyecto se requieren por el usuario nociones básicas de programación, electrónica, máquinas eléctricas y control.

1.2. Seguridad

Para la implementación en cualquier sistema del proyecto, se deberá respetar las normas y medidas de seguridad.

La fuente de alimentación del prototipo está preparada con dispositivos de seguridad eléctricos y/o mecánicos adecuados para la protección de los usuarios y de la máquina misma. Por lo tanto, está absolutamente prohibido alterar o quitar dichos dispositivos.

Es necesario controlar periódicamente que dichos dispositivos de seguridad (fusibles, limitadores de corriente, etc.) instalados en la fuente funcionen correctamente.

Una implementación errónea puede causar daños personales o materiales. Se declina cualquier responsabilidad que derive de alteraciones de estos dispositivos o la falta de utilización de los mismos, así como del uso irresponsable del sistema.

Se ha tenido en cuenta en el desarrollo del trabajo la siguiente normativa que debe ser de cumplimiento para cualquier variación, reproducción o implementación del proyecto:

- Real Decreto 1580/2006, de 22 de diciembre, por el que se regula la compatibilidad electromagnética de los equipos eléctricos y electrónicos. El presente real decreto regula la compatibilidad electromagnética de los equipos eléctricos y electrónicos que puedan crear perturbaciones electromagnéticas, o cuyo normal funcionamiento pueda verse perjudicado por dichas perturbaciones, exigiendo que cumplan un nivel adecuado de compatibilidad electromagnética a fin de garantizar el funcionamiento del mercado interior. (http://www.boe.es/diario_boe/txt.php?id=BOE-A-2007-973)
- Real Decreto 208/2005, de 25 de febrero, sobre aparatos eléctricos y electrónicos y la gestión de sus residuos. Este real decreto tiene por objeto, mediante la transposición de las Directivas 2002/95/CE del Parlamento

Europeo y del Consejo, de 27 de enero de 2003, sobre restricciones a la utilización de determinadas sustancias peligrosas en aparatos eléctricos y electrónicos, 2002/96/CE del Parlamento Europeo y del Consejo, de 27 de enero de 2003, sobre residuos de aparatos eléctricos y electrónicos, y 2003/108/CE del Parlamento Europeo y del Consejo, de 8 de diciembre de 2003, por la que se modifica la Directiva 2002/96/CE, establecer medidas para prevenir la generación de residuos procedentes de aparatos eléctricos y electrónicos y reducir su eliminación y la peligrosidad de sus componentes, así como regular su gestión para mejorar la protección del medio ambiente. (https://www.boe.es/diario_boe/txt.php?id=BOE-A-2005-3242)

- normas de seguridad para equipos eléctricos de medida, control y uso en laboratorio, así como drivers de potencia sistemas eléctricos: IEC 61010-1, EN 61010-1 y UL 61010-1, CSA 61010-1, UNE-EN 60745 , UNE-EN 61800.

1.3. Utilización

El sistema debe estar destinado a la utilización para la que ha sido diseñado y operando dentro de los márgenes de seguridad establecidos en las hojas de características del anexo [ANEXO 2]; cualquier otro tipo de utilización debe considerarse inadecuado y, por lo tanto, peligroso.

Cualquier variación del hardware o software respecto a los valores aportados debe estudiarse mediante simulación para prever su comportamiento.

Debe considerarse, al tratarse el prototipo de un sistema de eje móvil, la necesidad de despejar la zona dentro del rango de movimiento del elemento móvil para evitar daños materiales o personales.

1.4. Mantenimiento

Antes de iniciar cualquier tipo de intervención de mantenimiento, el sistema debe estar predispuesto de la siguiente manera:

-Desconectar la alimentación eléctrica.

-Para la sustitución de escobillas, utilizar solo aquellas homologadas o recomendadas por el fabricante.

-Al final de cada intervención que comporte el desmontaje de las protecciones, restablecer estas últimas en la posición correcta y asegurar que funcionen en modo correcto.

-Comprobar antes de la puesta en funcionamiento tras intervención el correcto cableado para evitar daños en el microcontrolador y el motor.

3. Condiciones particulares

3.1. Técnicas

3.1.1. Microcontrolador

Características:

- Las características técnicas del microcontrolador se encuentran en los anexos [ANEXO A2.5].

- *Firmware* versión: STM32Cube_FW_F4_V1.11.0

- *Driver*:

    - BSP

    - CMSIS

    - STM32F4xx_HAL_Driver

- Este producto está diseñado para cumplir con los requisitos de las siguientes normas de seguridad para equipos eléctricos de medida, control y uso en laboratorio:

- IEC 61010-1, EN 61010-1
- UL 61010-1, CSA 61010-1
- UNE-EN 60745
- UNE-EN 61800.

- El dispositivo SMT32F429-Discovery no está certificado para uso en lugares peligrosos.

Control de calidad.

- Debe comprobarse la serigrafía del procesador de la placa para cerciorarse de trabajar con la versión correcta.

- Es indispensable para una correcta comunicación con el resto de dispositivos, comprobar la calidad de los cables y conectores, especialmente el cable USB para utilizar el depurador de la tarjeta.

Condicionantes de la ejecución

Para la ejecución de la aplicación desarrollada, se deberá tener en cuenta los siguientes condicionantes:

- Instalación previa del software de programación de Keil uVision 5.
- Instalación previa del firmware de la tarjeta STM32F429-Discovery en la versión antes indicada mediante el software STMxCube.
- Instalación de los drivers necesarios para la depuración.
- Creación de una unidad virtual con el firmware a la que se encuentran redireccionadas las rutas de inclusión del proyecto. Para ello:

-Ve a "Inicio->Todos los programas->Inicio" y haz *click*-derecho en "Inicio" para seleccionar "Abrir".

-En la carpeta abierta, *click*-derecho y selecciona "Nuevo->Acceso directo"

-En el cuadro de diálogo abierto escribir "cmd" en la línea de comando y presionar "Enter".

-Hacer click-derecho en el icono creado y seleccionar "Propiedades".

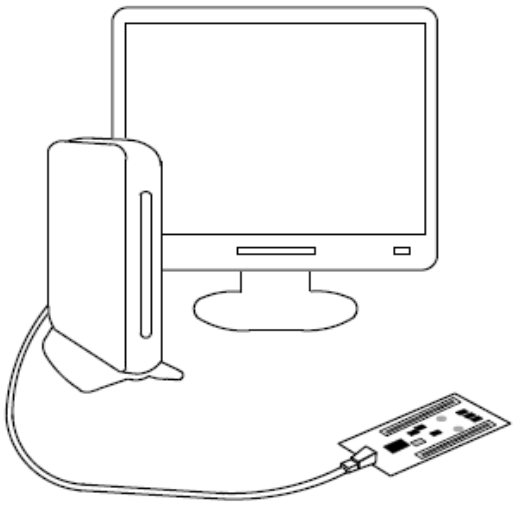
-En el campo "Destino" añadir el siguiente texto: %windir%\system32\cmd.exe /c "subst O: C:\STMicroelectronics\STM32Cube-Repository\STM32Cube_FW_F4_V1.9.0". Donde C:\... debe replazarse con la dirección en tu propio ordenador.

-Pinchar en el nuevo icono. Esto debería crear la unidad "O:\"

-Comprobar la correcta creación de la nueva unidad de disco "O:" abriéndola en el explorador de archivos.

Si tu "O:" esta ocupada, puedes adaptar la dirección. Comprueba el contenido de los archivos "MDK-ARM/Project.uvoptx" y "MDK-ARM/Project.uvprojx" para encontrar y reemplazar "O:\" con la dirección deseada.

Los requisitos mínimos del sistema de microcontrolador son:



Hardware requirements:
- USB cable type A to mini-B
- Computer with Windows XP, Vista or 7

Development toolchains:
- Altium TASKING VX-Toolset
- Atollic TrueSTUDIO
- IAR EWARM
- Keil MDK-ARM

Figura 1. Requerimientos mínimos del sistema microcontrolador

Para las conexiones necesarias entre el microcontrolador y el resto de elementos se comprobará el plano correspondiente (Plano 1) en el apartado de planos del presente proyecto.

3.1.2. Etapa de potencia

Características:

Las características técnicas la etapa de potencia se encuentran en los anexos [ANEXO A2.4].

Control de calidad.

- Debe comprobarse la serigrafía de la placa para comprobar que se tiene el componente indicado.

- Una inspección de funcionamiento correcto puede realizarse comprobando que la salida del regulador de tensión (indicada en la placa como "+5V") tiene 5V cuando se alimenta la placa en la entrada indicada como "+12V".

- Para las conexiones necesarias entre la etapa de potencia y el resto de elementos se comprobará el plano correspondiente (Plano 1) en el apartado de planos del presente proyecto.

3.1.3. Motor, bloque reductor y *encoder*

Características:

Las características técnicas del motor, el bloque reductor y el encoder se encuentran en los anexos [ANEXO A2.1., A2.2. y A2.3.].

Control de calidad.

- Comprobar el estado del motor y el giro suave del eje sin restricciones sin alimentación.

- Comprobar el giro tras alimentar a una tensión controlada con el eje libre de carga.

- Comprobar la salida del encoder en un osciloscopio.

Para las conexiones necesarias entre el conjunto de motor y encoder y el resto de elementos se comprobará el plano correspondiente (Plano 1) en el apartado de planos del presente proyecto.

3.1.4. Estructura

Características:

La dimensiones y cantidades de los distintos elementos se aprecia en los planos en al aparado correspondiente.

La estructura se compone de piezas de PLA impreso en una PRUSA i3 con altura de capa 0.2 mm y relleno 20%, con los archivos .stl obtenidos tras el diseño.

Existen tres tipos de varilla, todas ellas de acero cincado (8.8) y estándar din-975:

- Las varillas roscadas son de rosca métrica y de M10 y M8.

- Las varillas lisas son rectificadas y de diámetro 8 mm.

La unión entre las piezas impresas y las varillas que las conectas se realiza mediante tuercas hexagonales de rosca métrica de acero inoxidable a-2 según el estándar DIN-934, cada una de su respectiva métrica. La protección de la pieza se procura mediante arandelas del mismo material y estándar DIN-129-1 b, también en sus medidas correspondientes.

Se emplea para la transmisión del movimiento correa dentada del modelo GT2 en caucho y con nervios de hilo de acero, a modo de refuerzo.

Se utilizan para el deslizamiento, tanto de la correa como del carro, rodamientos de bolas. Para el movimiento lineal por las guías que suponen las varillas lisas se utilizan rodamientos lineales modelo LM8uu. Para el movimiento de la correa a través del tensor se utilizan dos rodamientos f623zz enfrentados, de manera que el bisel de los extremos evite el roce entre la pieza impresa y la correa.

Para la regulación de tensión de la correa en la pieza del tensor se utiliza un tornillo de M3 de 12 mm de largo con cabeza cilíndrica; y una tuerca de M3. Ambos del estándar DIN y de acero cincado a-3.

Para la sujeción del soporte de la correa al carro, se utilizan dos tornillos de M3 de 20 mm de largo con cabeza cilíndrica; y dos tuercas de M3. Todos del estándar DIN y de acero cincado a-3.

La sujeción de los rodamientos lineales al carro y de las varillas lisas a las esquinas se realiza mediante bridas de 100 mm de largo y 3 mm de ancho de plástico.

Por último las piezas que hacen de carro y lateral son de metacrilato de 8mm de espesor y han sido mecanizadas mediante corte láser a partir del diseño, y posteriormente taladradas para permitir su sujeción.

Control de calidad.

El control de calidad se realiza de forma visual, comprobando el correcto estado y dimensiones de los distintos elementos mediante un calibre.

3.2. Legales

Licencia:

El trabajo se publica bajo licencia GNU GPLv3 que permite cualquier modificación del proyecto siempre y cuando estas se realicen y publiquen bajo las mismas condiciones y se atribuya el trabajo al legítimo autor del mismo.

Esta es una licencia de software abierto, que permite el uso y disfrute del mismo en post de la mejora y desarrollo, sirviendo así el presente proyecto a su fin último.

No obstante si bien se permite y fomenta el uso y desarrollo del proyecto, el autor no se responsabiliza del uso distinto al cual se ha desarrollado que es el académico.

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR PROGRAMABLE DE PATRONES DE MOVIMIENTO PARA MOTORES DE CORRIENTE CONTINUA CON APLICACIÓN EN ROBOTS Y MÁQUINAS CNC

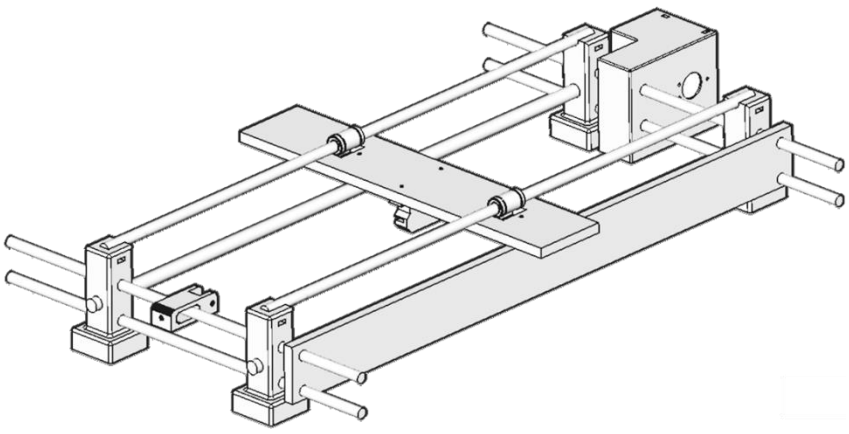
4.PRESUPUESTO

Autor:
D. José Félix González Rojo

Tutor:
D. Ángel Perles Ivars

Cotutor:
D. Miguel Sánchez López

Valencia, julio de 2016



Contenido del presupuesto

1. Sobre el presupuesto 2

2. Materiales 2

2.1. Materias primas 2

2.2. Productos industriales 3

3. Mano de obra 4

4. Coste del proyecto 4

1. Sobre el presupuesto

Para elaborar el presupuesto, los precios de la lista de materiales (materias primas y productos industriales) se han tomado de los catálogos proporcionados por los fabricantes y distribuidores de cada material.

Los costes de mano de obra se han calculado a partir de una estimación de precio de la labor de un graduado en ingeniería industrial (50€/h) y la duración en horas estimada del trabajo, siendo este de 12 ECTS y correspondiendo un ECTS a 25 horas de trabajo.

2. Materiales

2.1. Materias primas

MATERIALES				
Uds	Denominación	Cantidad	Precio(€)	Total
kg	Pieza Esquina	0,10	10,00	1,00
kg	Pieza Tensor correa	0,02	10,00	0,20
kg	Pieza Soporte correa	0,01	10,00	0,10
kg	Pieza Soporte motor	0,07	10,00	0,70
kg	Pieza Carro	0,05	12,00	0,60
kg	Pieza Lateral	0,08	12,00	0,96
m	Varillas lisas de acero	0,80	2,77	2,22
m	Varillas roscadas de acero M8	1,40	0,37	0,52
m	Varillas roscadas de acero M10	0,90	0,58	0,52
m	Correa GT2	0,80	1,42	1,13
Subtotal Materias Primas				**7,95**

Tabla 1 Materiales. Materias Primas

2.2. Productos industriales

MATERIALES				
Uds	Denominación	Cantidad	Precio(€)	Total
u	Rodamiento f623zz	2	0,370	0,750
u	Rodamiento LM8uu	2	0,710	1,414
u	Tuercas M8	38	0,009	0,334
u	Tuercas M10	8	0,020	0,159
u	Tuercas M3	3	0,0013	0,004
u	Arandela M8	38	0,0055	0,209
u	Arandela M10	8	0,0107	0,086
u	Tornillo M3	3	0,0096	0,029
u	Brida 3x15	6	0,0126	0,075
u	Poleas GT2	1	0,80	0,80
u	Resistencia de potencia 0.5 ohm 4W	1	1,89	1,89
u	Cable Dupont 20 cm	11	0,028	0,308
u	Motor corriente continua con reductora y Encoder	1	222,00	222,00
u	Placa L298	1	1,97	1,97
u	STM32F429-Discovery	1	30,2	30,2
	Subtotal Productos Industriales			**260.226**

Tabla 2 Materiales. Productos Industriales

3. Mano de obra

MANO DE OBRA				
Uds	Denominación	Cantidad	Precio(€)	Total
h	Diseño y programación	300	50,00	15.000,00
	Graduado en Ingeniería electronica industrial y automática			
	SUBTOTAL MANO DE OBRA			**15.000,00**

Tabla 3. Mano de obra

4. Coste del proyecto

COSTE DEL PROYECTO		
Materia prima	7,95	
Productos industriales	260,226	
Mano de obra	15.000,00	
TOTAL	**15.268,176 €**	
IVA	21%	3.206,317
Gastos adicionales	10%	1.526,817

Tabla 4. Coste del proyecto

PVP	**20.001,31 €**

El precio total es de veinte mil un euros con treinta y un céntimos de euro.

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR PROGRAMABLE DE PATRONES DE MOVIMIENTO PARA MOTORES DE CORRIENTE CONTINUA CON APLICACIÓN EN ROBOTS Y MÁQUINAS CNC

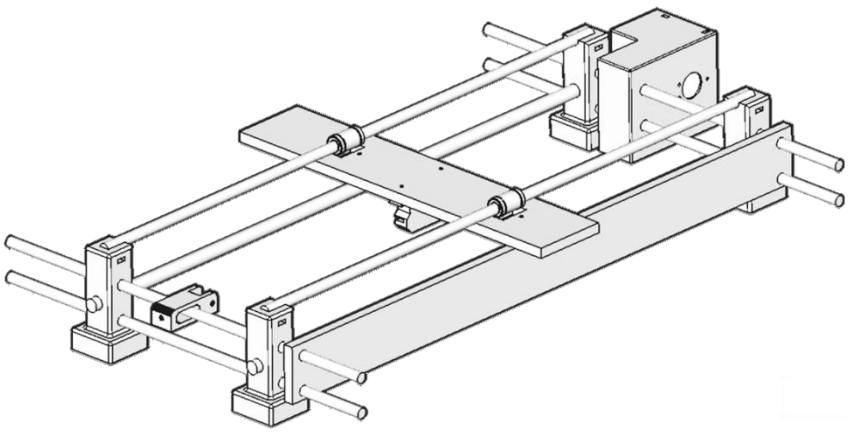
5.ANEXOS

Autor:
D. José Félix González Rojo

Tutor:
D. Ángel Perles Ivars

Coutor:
D. Miguel Sánchez López

Valencia, julio de 2016



Contenido Anexo 1. Código

A1.1. Código de identificación del modelo. Modelado no paramétrico 3

1.1. Identificación del modelo de posición del motor ante entrada de tensión 3

1.2. Identificación del modelo de velocidad del motor ante entrada de tensión 5

1.3. Script de Matlab para la obtención de gráficas a partir de los datos de posición 7

1.4. Script de Matlab para la obtención de gráficas a partir de los datos de velocidad 7

1.5. Script de Matlab para la obtención de los modelos discretos a partir de las funciones de transferencia de los continuos 8

A1.2. Código de identificación del modelo. Modelado paramétrico 8

2.1. Identificación del modelo de posición del motor ante entrada de tensión mediante señal PRBS 8

2.2. Identificación del modelo de velocidad del motor ante entrada de tensión mediante señal PRBS 10

2.3. Script de Matlab para la obtención de gráficas a partir de los datos de posición 12

2.4. Script de Matlab para la obtención de gráficas a partir de los datos de velocidad 13

A1.3. Código de identificación e implementación de un control PID 14

3.1. Diseño y simulación del control tipo PID para el modelo de velocidad 14

3.2. Diseño y simulación del control tipo PID para el modelo de posición 17

3.3. Script para la obtención de las gráficas con los datos de la implementación de velocidad y los errores 20

3.4. Script para la obtención de las gráficas con los datos de la implementación de posición y los errores 20

A1.4. Código de implementación de los controladores y adaptación de la acción de control 21

4.1. control.c 21

4.2. control.h 25

A1.5. Código de configuración del reloj de la tarjeta e inicialización 26

5.1. Configuración del reloj 26

5.2. Programa principal con inicialización. main.c 27

A1.6. Código de configuración del contador que lleva el tiempo de muestreo 28

6.1. Funciones de control de HAL 28

6.2. Implementación para control del muestreo 30

A1.7. Código de configuración, inicialización y manejo del encoder óptico incremental ... 31

7.1. Configuración del encoder. encoder.c 31

7.2. Configuración del encoder. encoder.h 33

7.3. Ejemplo de uso, medida de posición y velocidad 33

A1.8. Código de configuración, inicialización y manejo del ADC 34

8.1. Configuración del ADC. adc.c 34

8.2. Configuración del ADC. adc.h 38
8.3. Ejemplo de uso midiendo la corriente del motor 38
A1.9. Código de configuración, inicialización y manejo del generador de señal PWM 40
9.1. Configuración del generador de señal PWM. pwm.c 40
9.2. Configuración del generador de señal PWM. pwm.h 43
9.3. Ejemplo de uso generando una señal en dos canales 44
A1.10. Código de generación de una señal PRBS para PWM 45
10.1. prbs.c 45
10.2. prbs.h 47
A1.11. Código de generación de referencias mediante perfiles de movimiento 47
11.1. refgentray.c 47
11.2. refgentray.h 53
11.3. Script de función de Matlab para generar una trayectoria de tercer orden para simulaciones 53

A1.1. Código de identificación del modelo. Modelado no paramétrico

1.1. Identificación del modelo de posición del motor ante entrada de tensión

```
/******************************************************************
* @file    ModeloPos_main.c
  * @author  Jose Felix Gonzalez Rojo
  * @version V1.0
  * @date    June-2016
  * @brief   Main program body
******************************************************************/

/* Includes ------------------------------------------------------*/
#include "main.h"
#include "math.h"
#include "stdio.h"
#include "stdlib.h"
#include "stm32f4xx_hal.h"
#include "encoder.h"
#include "pwm.h"

/* Private function prototypes -----------------------------------*/

static void SystemClock_Config(void);
static void Error_Handler(void);

/**
  * @brief  Main program
  * @param  None
  * @retval None
  */

/* Main Function */
int main(void)
{
      /* Reset of all peripherals, Initializes the Flash interface and
      the Systick. */
      HAL_Init();
      /* Configure the system clock to 180 MHz */
       SystemClock_Config();

      /* Variable Declaration */
      uint32_t encoder_position = 0;
      uint32_t encoder_old_position = 0;
      float speed_value = 0;
      uint32_t position[2000];
      uint32_t old_position[2000];
      float speed_data[2000];
      float Input[2000];
      float duty1=0;
      uint32_t Tinit = 0;
      uint32_t Tend = 0;

      /* Initialize all configured peripherals */
      encoder_Init();
      Pwm_Init();
      pwm_Set(TIM_CHANNEL_1, 0); //holding motor to brake
```

ANEXO 1. CÓDIGO

```
        pwm_Set(TIM_CHANNEL_2, 0);
        TIM1->CNT=0; //setting encoder value to 0
         while (i<2000)
        {
              // control process start
              Tinit = 0;
              Tend = 0;
              Tinit = HAL_GetTick();//starting time


              // Sensor read. Getting current value of pos and vel
              encoder_position= encoder_Read();
              position[i]= encoder_position;
              speed_value =(encoder_position -
encoder_old_position)*(10.0F);
              speed_data[i]= speed_value;

              // Calculing control action
              Duty1=0.25 //change when needed

              Input[i]=duty1;
              // Aplaying control
              pwm_Set(TIM_CHANNEL_1, duty1);
              pwm_Set(TIM_CHANNEL_2, 0);

              // Variable update
              i++;
              encoder_old_position = encoder_Read();
              old_position[i]= encoder_old_position;

              // Control process end
              Tend = HAL_GetTick(); //ending time
              // Wait for period end
              if (Tend-Tinit < 10)
              {
                    HAL_Delay(10-(Tend-Tinit)); //wait the period
              }
              else
              {
                    printf("Error! Too long procesing time \n");
                    Error_Handler();
              }
        }

        for (uint16_t j=0; j<2000; j++)
        {
        printf("%d %d %f %f %d %d\n", position[j], old_position[j],
  speed_data[j],speed_value, Input[i],j,j*10);
        }
        While(1)//avoids hardfault error
        {
        }
  }

  /************* (C) COPYRIGHT STMicroelectronics *****END OF FILE*****/
```

1.2. Identificación del modelo de velocidad del motor ante entrada de tensión

```
/**********************************************************************
* @file    ModeloPos_main.c
  * @author  Jose Felix Gonzalez Rojo
  * @version V1.0
  * @date    June-2016
  * @brief   Main program body
**********************************************************************/

/* Includes ------------------------------------------------------------*/
#include "main.h"
#include "math.h"
#include "stdio.h"
#include "stdlib.h"
#include "stm32f4xx_hal.h"
#include "encoder.h"
#include "pwm.h"

/* Private function prototypes -----------------------------------------*/

static void SystemClock_Config(void);
static void Error_Handler(void);

/**
  * @brief  Main program
  * @param  None
  * @retval None
  */

/* Main Function */
int main(void)
{
      /* Reset of all peripherals, Initializes the Flash interface and
      the Systick. */
      HAL_Init();
      /* Configure the system clock to 180 MHz */
       SystemClock_Config();

      /* Variable Declaration */
      uint32_t encoder_position = 0;
      uint32_t encoder_old_position = 0;
      float speed_value = 0;
      uint32_t position[2000];
      uint32_t old_position[2000];
      float speed_data[2000];
      float Input[2000];
      float duty1=0;
      uint32_t Tinit = 0;
      uint32_t Tend = 0;

      /* Initialize all configured peripherals */
      encoder_Init();
      Pwm_Init();
      pwm_Set(TIM_CHANNEL_1, 0); //holding motor to brake
      pwm_Set(TIM_CHANNEL_2, 0);
      TIM1->CNT=0; //setting encoder value to 0
       while (i<2000)
```

ANEXO 1. CÓDIGO

```
        {
              // control process start
              Tinit = 0;
              Tend = 0;
              Tinit = HAL_GetTick(); //starting time


              // Sensor read. Getting current value of pos and vel
              encoder_position= encoder_Read();
              position[i]= encoder_position;
              speed_value =(encoder_position -
encoder_old_position)*(10.0F);
              speed_data[i]= speed_value;

              // Calculing control action
              if (i<500) //avoiding dead zone
              {
                    duty1=0.15
              }
              else
              {
                    Duty1=0.25 //change when needed
              }
              Input[i]=duty1;
              // Aplaying control
              pwm_Set(TIM_CHANNEL_1, duty1);
              pwm_Set(TIM_CHANNEL_2, 0);

              // Variable update
              i++;
              encoder_old_position = encoder_Read();
              old_position[i]= encoder_old_position;

              // Control process end
              Tend = HAL_GetTick(); //ending time
              // Wait for period end
              if (Tend-Tinit < 10)
              {
                    HAL_Delay(10-(Tend-Tinit));//wait the period
              }
              else
              {
                    printf("Error! Too long procesing time \n");
                    Error_Handler();
              }
        }

        for (uint16_t j=0; j<2000; j++)
        {
        printf("%d %d %f %f %d %d\n", position[j], old_position[j],
  speed_data[j],speed_value, Input[i],j,j*10);
        }
        While(1)//avoids hardfault error
        {
        }
  }

  /************* (C) COPYRIGHT STMicroelectronics *****END OF FILE*****/
```

1.3. Script de Matlab para la obtención de gráficas a partir de los datos de posición

```
datos=load('testp100.txt')

figure

plot(datos(1:500,6),datos(1:500,1),'b')
hold on
stairs(datos(1:500,6),datos(1:500,4),'k')
xlabel('Tiempo (ms)')
ylabel('Accion/Posicion')
legend('Posicion motor (pulsos o grados)','Accion (%)')
title('Respuesta pos. 100')
grid

%Checking continous model

figure
plot(datos(1:500,6),datos(1:500,1),'b')
hold on
stairs(datos(1:500,6),datos(1:500,4),'k')
hold on
plot(datos(1:500,6),posval1(1:500,1),'k')
hold on
plot(datos(1:500,6),posval1d(1:500,1),'r')
xlabel('Tiempo (ms)')
ylabel('Accion/Posicion')
legend('Posicion motor (pulsos o grados)','Accion (%)','Resp.
Modelo','Resp. Modelo Discr.')
title('Respuesta pos. 100')
grid
```

1.4. Script de Matlab para la obtención de gráficas a partir de los datos de velocidad

```
datos=load('datosvel100.txt')
figure

plot(datos(:,6),1000*datos(:,3),'b')
hold on
stairs(datos(:,6),datos(:,4),'k')
xlabel('Tiempo (ms)')
ylabel('Accion/Velocidad')
legend('Velocidad motor (pulsos/s o grados/s)','Accion (%)')
title('Respuesta vel. 75')
grid

%Checking continous model

figure
plot(datos(:,6),1000*datos(:,3),'b')
hold on
stairs(datos(:,6),datos(:,4),'k')
hold on
plot(datos(:,6),velval1(1:500,1),'k')
hold on
```

```
plot(datos(:,6),velval1d(1:500,1),'r')
xlabel('Tiempo (ms)')
ylabel('Accion/Velocidad')
legend('Velocidad motor (pulsos/s o grados/s)','Accion (%)','Resp.
Modelo','Resp. Modelo Discr.')
title('Respuesta vel. 100')
grid
```

1.5. Script de Matlab para la obtención de los modelos discretos a partir de las funciones de transferencia de los continuos

```
numpos=7.56
numvel=7.4;
denpos=[0.075 1 0]
denvel=[0.075 1];

[numpd, denpd]=c2dm(numpos,denpos,0.01,'zoh')
[numvd, denvd]=c2dm(numvel,denvel,0.01,'zoh')
```

A1.2. Código de identificación del modelo. Modelado paramétrico

2.1. Identificación del modelo de posición del motor ante entrada de tensión mediante señal PRBS

```
/**********************************************************************
* @file    ModeloPos_main.c
  * @author  Jose Felix Gonzalez Rojo
  * @version V1.0
  * @date    June-2016
  * @brief   Main program body
**********************************************************************/

/* Includes ------------------------------------------------------------*/
#include "main.h"
#include "math.h"
#include "stdio.h"
#include "stdlib.h"
#include "stm32f4xx_hal.h"
#include "encoder.h"
#include "pwm.h"
#include "PRBS.h"

/* Private function prototypes -----------------------------------------*/

static void SystemClock_Config(void);
static void Error_Handler(void);

/**
  * @brief  Main program
  * @param  None
  * @retval None
  */

/* Main Function */
int main(void)
{
```

```
	/* Reset of all peripherals, Initializes the Flash interface and
	the Systick. */
	HAL_Init();
	/* Configure the system clock to 180 MHz */
	 SystemClock_Config();

	/* Variable Declaration */
	uint32_t encoder_position = 0;
	uint32_t encoder_old_position = 0;
	float speed_value = 0;
	uint32_t position[2000];
	uint32_t old_position[2000];
	float speed_data[2000];
	float Input[2000];
	float duty1=0;
	uint32_t Tinit = 0;
	uint32_t Tend = 0;

	/* Initialize all configured peripherals */
	encoder_Init();
	Pwm_Init();
	pwm_Set(TIM_CHANNEL_1, 0); //holding motor to brake
	pwm_Set(TIM_CHANNEL_2, 0);
	TIM1->CNT=0; //setting encoder value to 0
	 while (i<2000)
	{
		// control process start
		Tinit = 0;
		Tend = 0;
		Tinit = HAL_GetTick();//starting time


		// Sensor read. Getting current value of pos and vel
		encoder_position= encoder_Read();
		position[i]= encoder_position;
		speed_value =(encoder_position -
encoder_old_position)*(10.0F);
		speed_data[i]= speed_value;

		// Calculing control action
		Duty1=PWM_duty_prbs(0.5,0.01,3);
		Input[i]=duty1;

		// Aplaying control
		pwm_Set(TIM_CHANNEL_1, duty1);
		pwm_Set(TIM_CHANNEL_2, 0);

		// Variable update
		i++;
		encoder_old_position = encoder_Read();
		old_position[i]= encoder_old_position;

		// Control process end
		Tend = HAL_GetTick(); //ending time
		// Wait for period end
		if (Tend-Tinit < 10)
		{
			HAL_Delay(10-(Tend-Tinit)); //wait the period
		}
		else
```

```
            {
                  printf("Error! Too long procesing time \n");
                  Error_Handler();
            }
      }

      for (uint16_t j=0; j<2000; j++)
      {
      printf("%d %d %f %f %d %d\n", position[j], old_position[j],
speed_data[j],speed_value, Input[i],j,j*10);
      }
      While(1)//avoids hardfault error
      {
      }
}

/************* (C) COPYRIGHT STMicroelectronics *****END OF FILE*****/
```

2.2. Identificación del modelo de velocidad del motor ante entrada de tensión mediante señal PRBS

```
/**************************************************************************
* @file    ModeloPos_main.c
  * @author  Jose Felix Gonzalez Rojo
  * @version V1.0
  * @date    June-2016
  * @brief   Main program body
**************************************************************************/

/* Includes ------------------------------------------------------------*/
#include "main.h"
#include "math.h"
#include "stdio.h"
#include "stdlib.h"
#include "stm32f4xx_hal.h"
#include "encoder.h"
#include "pwm.h"
#include "PRBS.h"


/* Private function prototypes -----------------------------------------*/

static void SystemClock_Config(void);
static void Error_Handler(void);

/**
  * @brief  Main program
  * @param  None
  * @retval None
  */

/* Main Function */
int main(void)
{
      /* Reset of all peripherals, Initializes the Flash interface and
      the Systick. */
      HAL_Init();
      /* Configure the system clock to 180 MHz */
       SystemClock_Config();
```

```
        /* Variable Declaration */
        uint32_t encoder_position = 0;
        uint32_t encoder_old_position = 0;
        float speed_value = 0;
        uint32_t position[2000];
        uint32_t old_position[2000];
        float speed_data[2000];
        float Input[2000];
        float duty1=0;
        uint32_t Tinit = 0;
        uint32_t Tend = 0;

        /* Initialize all configured peripherals */
        encoder_Init();
        Pwm_Init();
        pwm_Set(TIM_CHANNEL_1, 0); //holding motor to brake
        pwm_Set(TIM_CHANNEL_2, 0);
        TIM1->CNT=0; //setting encoder value to 0
         while (i<2000)
        {
              // control process start
              Tinit = 0;
              Tend = 0;
              Tinit = HAL_GetTick(); //starting time


              // Sensor read. Getting current value of pos and vel
              encoder_position= encoder_Read();
              position[i]= encoder_position;
              speed_value =(encoder_position -
encoder_old_position)*(10.0F);
              speed_data[i]= speed_value;

              // Calculing control action
              if (i<500) //avoiding dead zone
              {
                    duty1=0.15
              }
              else
              {
                    Duty1= PWM_duty_prbs(0.5,0.01,3);
              }
              Input[i]=duty1;
              // Aplaying control
              pwm_Set(TIM_CHANNEL_1, duty1);
              pwm_Set(TIM_CHANNEL_2, 0);

              // Variable update
              i++;
              encoder_old_position = encoder_Read();
              old_position[i]= encoder_old_position;

              // Control process end
              Tend = HAL_GetTick(); //ending time
              // Wait for period end
              if (Tend-Tinit < 10)
              {
                    HAL_Delay(10-(Tend-Tinit));//wait the period
              }
```

```
            else
            {
                  printf("Error! Too long procesing time \n");
                  Error_Handler();
            }
      }

      for (uint16_t j=0; j<2000; j++)
      {
      printf("%d %d %f %f %d %d\n", position[j], old_position[j],
speed_data[j],speed_value, Input[i],j,j*10);
      }
      While(1)//avoids hardfault error
      {
      }
}

/************* (C) COPYRIGHT STMicroelectronics *****END OF FILE*****/
```

2.3. Script de Matlab para la obtención de gráficas a partir de los datos de posición

```
data=load('testpprbs.txt')

pos=data(:,1);
pos_grad=pos*(6.283185/360);
u=data(:,4);
it=data(:,6);

Yk = pos(5:1900)*-1; %Grados
Yk_1 = pos(4:1899)*-1; %Grados
%Yk = pos_grad(5:1900)*-1; %Radianes
%Yk_1 = pos_grad(4:1899)*-1; %Radianes
Uk = u(6-4:1901-4);
Uk_1 = u(1:1900-4);
fi =horzcat(Yk, Yk_1, Uk, Uk_1);

Yk1 = pos(6:1901)
%Yk1 = pos_grad(6:1901)

theta = inv(fi'*fi)*fi'*Yk1

mp=[0:1:1900-5]'
Ea=horzcat(mp,Uk)
Ya=horzcat(mp,Yk1)

data2=load('testp50.txt')

figure
plot(data2(:,6),data2(:,1),'b')
hold on
stairs(data2(:,6),data2(:,4),'k')
hold on
stairs(data2(:,6),posd(1:1903,1),'k')
xlabel('Tiempo (ms)')
ylabel('Accion/Velocidad')
legend('Posicion motor (pulsos o grados)','Accion (%)','Resp. Modelo
Discr.')
title('Respuesta pos. 50')
```

```
grid
%d=4
%Radianes
%a1=-0.8584;
%a2=-0.1416;
%b1=0.0008;
%b2=0.0008;

%Grados
%a1=-0.8584;
%a2=-0.1416;
%b1=0.0474;
%b2=0.0487;
```

2.4. Script de Matlab para la obtención de gráficas a partir de los datos de velocidad

```
data=load('testprbsrad2.txt')

pos=data(:,1);
pos_2=data(:,2);
vel=1000*data(:,3);
pos_grad=pos*(6.283185/360);
pos_2_grad=pos_2*(6.283185/360);
vel_grad=1000*(pos_grad-pos_2_grad)*10;
u=data(:,4);
it=data(:,6);

Yk = vel(521:1928)*-1; %Grados
%Yk = vel_grad(521:1928)*-1; %Radianes
Uk = u(522-4:1925);
fi =horzcat(Yk,Uk);

Yk1 = vel(522:1929)
%Yk1 = vel_grad(522:1928)
Yk1 = vertcat(Yk1,0)

theta = inv(fi'*fi)*fi'*Yk1

mp=[0:1:1928-521]'
Ea=horzcat(mp,Uk)
Ya=horzcat(mp,Yk1)

data2=load('datosvel50.txt')

figure
plot(data2(:,6),1000*data2(:,3),'b')
hold on
stairs(data2(:,6),data2(:,4),'k')
hold on
stairs(data(:,6),veld(:,1),'k')
xlabel('Tiempo (ms)')
ylabel('Accion/Velocidad')
legend('Velocidad motor (pulsos/s o grados/s)','Accion (%)','Resp.
Modelo Discr.')
title('Respuesta vel. 50')
grid
```

```
%Radianes
%a1=-0.0838;
%b0=10.4253;
%d=4
%Grados
%a1=-0.0838;
%b0=6.0272;
```

A1.3. Código de identificación e implementación de un control PID

3.1. Diseño y simulación del control tipo PID para el modelo de velocidad

```
close all;
clear all;
clc;

T=0.01 %Periodo

%Modelo del proceso
num= [7.4]
den= [0.075 1]
[numd,dend]=c2dm(num,den,0.01,'zoh')

%Parámetros de la respuesta escalón
step(tf(num,den));
title('Respuesta a un escalón BA');
grid
K= 7.4;
T1= 0.034; %tiempo salida 28.3% de K
T2= 0.086; %tiempo salida 63.2% de K
Tp=1.5*(T2-T1)
T0=T2-Tp

%% Control P
Tfinal=1; % final de la sim

% A partir de la respuesta escalón BA
Kp=Tp/(K*T0)
%Regulador
numreg=Kp
denreg=1
%Regulador ajustado
numreg=0.127
denreg=1
%Referencia
Referencia=800

sim('simPID.slx');
subplot(2,1,1);
plot(t,ref,'g');
hold on;
plot(t,y);
grid;
xlabel('Tiempo (s)');
ylabel('Salida');
```

```
title('Control P. Velocidad');
legend('Referencia','Respuesta');
subplot(2,1,2);
stairs(t,u,'r');
xlabel('Tiempo (s)');
ylabel('Acc. Control');
grid;
legend('Acc. Control')

CIEC=sum(e.^2)
CIEA=sum(abs(e))
%% Control PD
Tfinal=1; % final de la sim

% A partir de la respuesta escalón BA
Kpd=1.2*Tp/(K*T0)
Td=0.5*T0
q0=Kpd*(T+Td)/T
q1=-Kpd*Td/T
%Regulador
numreg=[q0 q1]
denreg=[1 0]
%Referencia
Referencia=800;

sim('simPID.slx');
figure;
subplot(2,1,1);
plot(t,ref,'g');
hold on;
plot(t,y);
xlabel('Tiempo (s)');
ylabel('Salida');
title('Control PD. Velocidad');
legend('Referencia','Respuesta');
grid;
subplot(2,1,2);
stairs(t,u,'r');
xlabel('Tiempo (s)');
ylabel('Acc. Control');
legend('Acc. Control');
grid;

CIEC=sum(e.^2)
CIEA=sum(abs(e))

%% Control PI
Tfinal=1; % final de la sim

% A partir de la respuesta escalón BA
Kpi=0.9*Tp/(K*T0)
Ti=T0/0.3
q0=Kpi
q1=Kpi*(T-Ti)/Ti
% Ajustado
Kpi=0.05
Ti=0.05
q0=Kpi
q1=Kpi*(T-Ti)/Ti
```

ANEXO 1. CÓDIGO

```
%Regulador
numreg=[q0 q1]
denreg=[1 -1]
%Referencia
Referencia=800;

sim('simPID.slx');
figure;
subplot(2,1,1);
plot(t,ref,'g');
hold on;
plot(t,y);
xlabel('Tiempo (s)');
ylabel('Salida');
title('Control PI. Velocidad');
legend('Referencia','Respuesta');
grid;
subplot(2,1,2);
stairs(t,u,'r');
xlabel('Tiempo (s)');
ylabel('Acc. Control');
legend('Acc. Control');
grid;

CIEC=sum(e.^2)
CIEA=sum(abs(e))

%% Control PID
Tfinal=1; % final de la sim

% A partir de la respuesta escalón BA
Kpid=1.2*Tp/(K*T0)
Ti=2*T0
Td=0.5*T0
q0=Kpid*(T+Td)/T
q1=Kpid*(-1+(T/Ti)-2*Td/T)
q2=Kpid*Td/T
%Regulador
numreg=[q0 q1 q2];
denreg=[1 -1 0];
%Referencia
Referencia=800;

sim('simPID.slx');
figure;
subplot(2,1,1);
plot(t,ref,'g');
hold on;
plot(t,y);
xlabel('Tiempo (s)');
ylabel('Salida');
title('Control PID. Velocidad');
legend('Referencia','Respuesta');
grid;
subplot(2,1,2);
stairs(t,u,'r');
xlabel('Tiempo (s)');
ylabel('Acc. Control');
legend('Acc. Control');
grid;
```

```
CIEC=sum(e.^2)
CIEA=sum(abs(e))
```

3.2. Diseño y simulación del control tipo PID para el modelo de posición

```
close all;
clear all;
clc;

T=0.01; %Periodo

%Modelo del proceso
num= [7.56];
den= [0.075 1 0];
[numd,dend]=c2dm(num,den,0.01,'zoh')

%Parámetros de la respuesta mantenida
Kc= 27.056;
Tc= 0.1;
figure;
step(feedback(tf(Kc*numd,dend,0.01),1));
grid;
title('Respuesta a un escalón BC');


%% Control P
Tfinal=5; %Tiempo final de la prueba

% de la respuesta sostenida del motor
Kp=0.5*Kc
%Regulador
numreg=Kp
denreg=1
%Referencia
Referencia=90;

sim('simControl.mdl');
figure;
subplot(2,1,1);
plot(t,ref,'g');
hold on;
plot(t,y);
xlabel('Tiempo (s)');
ylabel('Salida');
title('Control P. Posición');
legend('Referencia','Respuesta');
grid;
subplot(2,1,2);
stairs(t,u,'r');
xlabel('Tiempo (s)');
ylabel('Acc. Control');
legend('Acc. Control');
grid;
```

```
CIEC=sum(e.^2)
CIEA=sum(abs(e))

%% Control PD
Tfinal=5; %Tiempo final de la prueba

% de la respuesta sostenida del motor
Kpd=0.6*Kc
Td=Tc/8
q0=Kpd*(T+Td)/T
q1=-Kpd*Td/T
%Regulador
numreg=[q0 q1];
denreg=[1 0];
%Referencia
Referencia=90;

sim('simControl.mdl');
figure;
subplot(2,1,1);
plot(t,ref,'g');
hold on;
plot(t,y);
xlabel('Tiempo (s)');
ylabel('Salida');
title('Control PD. Posición');
legend('Referencia','Respuesta');
grid;
subplot(2,1,2);
stairs(t,u,'r');
xlabel('Tiempo (s)');
ylabel('Acc. Control');
legend('Acc. Control');
grid;
CIEC=sum(e.^2)
CIEA=sum(abs(e))


%% Control PI
Tfinal=5; %Tiempo final de la prueba

% de la respuesta sostenida del motor
Kpi=0.45*Kc
Ti=Tc/1.2
q0=Kpi
q1=Kpi*(T-Ti)/Ti
%Regulador
numreg=[q0 q1];
denreg=[1 -1];
%Referencia
Referencia=90;

sim('simControl.mdl');
figure;
subplot(2,1,1);
plot(t,ref,'g');
hold on;
plot(t,y);
xlabel('Tiempo (s)');
```

```
ylabel('Salida');
title('Control PI. Posición');
legend('Referencia','Respuesta');
grid;
subplot(2,1,2);
stairs(t,u,'r');
xlabel('Tiempo (s)');
ylabel('Acc. Control');
legend('Acc. Control');
grid;
CIEC=sum(e.^2)
CIEA=sum(abs(e))


%%Control PID
Tfinal=5; %Tiempo final de la prueba

% de la respuesta sostenida del motor
Kpid=0.6*Kc
Ti=Tc/1.2
Td=Tc/8
Kpid=2.9968
Ti=0.25
Td=0.02896
q0=Kpid*(T+Td)/T
q1=Kpid*(-1+(T/Ti)-2*Td/T)
q2=Kpid*Td/T
%Regulador
numreg=[q0 q1 q2];
denreg=[1 -1 0];
%Referencia
Referencia=90;

sim('simControl.mdl');
figure;
subplot(2,1,1);
plot(t,ref,'g');
hold on;
plot(t,y);
xlabel('Tiempo (s)');
ylabel('Salida');
title('Control PID. Posición');
legend('Referencia','Respuesta');
grid;
subplot(2,1,2);
stairs(t,u,'r');
xlabel('Tiempo (s)');
ylabel('Acc. Control');
legend('Acc. Control');
grid;
CIEC=sum(e.^2)
CIEA=sum(abs(e))
```

3.3. Script para la obtención de las gráficas con los datos de la implementación de velocidad y los errores

```
Datos=load('testPI2.txt');

figure
subplot(2,2,1)
plot(Datos(:,2), Datos(:,3), 'r')
hold on
plot(Datos(:,2), Datos(:,4), 'b')
grid
xlabel('Tiempo (ms)');
ylabel('Vel.motor(°/s)');
legend('Referencia','Velocidad');
title('Velocidad motor con acción de control PI');
subplot(2,2,2)
plot(Datos(:,2), Datos(:,6), 'r')
grid
xlabel('Tiempo (ms)');
ylabel('Valor');
legend('Error Cuad.');
title('CEIC');
subplot(2,2,3)
plot(Datos(:,2), Datos(:,5), 'r')
grid
xlabel('Tiempo (ms)');
ylabel('%');
legend('Acc. Control');
title('Acc. Control');
subplot(2,2,4)
plot(Datos(:,2), Datos(:,7), 'r')
grid
xlabel('Tiempo (ms)');
ylabel('Valor');
legend('Error Abs.');
title('CEIA');
```

3.4. Script para la obtención de las gráficas con los datos de la implementación de posición y los errores

```
Datos=load('testPIDspl3.txt');

figure
subplot(2,2,1)
plot(Datos(:,2), Datos(:,3), 'r')
hold on
plot(Datos(:,2), Datos(:,4), 'b')
grid
xlabel('Tiempo (ms)');
ylabel('Pos.motor (°)');
legend('Referencia','Posicion');
title('Posicion motor con acción de control PID');
subplot(2,2,2)
plot(Datos(:,2), Datos(:,6), 'r')
grid
xlabel('Tiempo (ms)');
ylabel('Valor');
legend('Error Cuad.');
title('CEIC');
```

```
subplot(2,2,3)
plot(Datos(:,2), Datos(:,5), 'r')
grid
xlabel('Tiempo (ms)');
ylabel('%');
legend('Acc. Control');
title('Acc. Control');
subplot(2,2,4)
plot(Datos(:,2), Datos(:,7), 'r')
grid
xlabel('Tiempo (ms)');
ylabel('Valor');
legend('Error Abs.');
title('CEIA');
```

A1.4. Código de implementación de los controladores y adaptación de la acción de control

4.1. control.c

```
/**
    @file control.c
    @brief xxx

      Required resources:

    @author Jose Felix
    @date May 2016
*/

#include "control.h"

/*****************************************************************
***************************/
/**
  @brief float control_PI (float ref,float value)
           Gets control action for reaching the current
           reference.

  Example:
  @verbatim
      AC=control_PI(target_speed,speed_value);

  @endverbatim
**/

float control_PI (float ref, float value)
{
      static float q0;
      static float q1;
      static float e; //current error
      static float e1; //previous error
      static float u; //current control action
      static float u1; //previous control action

      q0=0.135F;  //controler parameters init
  q1=-0.0928F;
```

```
      e= ref-value; //calculate error
      u=q0*e+q1*e1+u1; //calculates control action

      //varaible update

      if((u >= -100.0)&&(u <= 100.0)) //antiwindup
      {
  e1=e;
      }

  u1=u;


      //control action saturation (working on +-10V here)
      if(u > 100.0F)
      {
            u=100.0F;
      }
      else if(u < -100.0F)
      {
            u=-100.0F;
      }

      return(u);
}

/*****************************************************************
****************************/
/**
  @brief float control_PD (float ref,float value)
            Gets control action for reaching the current
            reference.

  Example:
  @verbatim
      AC=control_PD(target_speed,pos_value);

  @endverbatim
**/

float control_PD (float ref, float value)
{
      static float q0;
      static float q1;
      static float e; //current error
      static float e1; //previous error
      static float u; //current control action

      q0=0.135F;  //controler parameters init
  q1=-0.0928F;


      e= ref-value; //calculate error
      u=q0*e+q1*e1; //calculates control action

      //varaible update

  e1=e;
```

```
      //control action saturation (working on +-10V here)
      if(u > 100.0F)
      {
            u=100.0F;
      }
      else if(u < -100.0F)
      {
            u=-100.0F;
      }

      return(u);
}

/*****************************************************************
***************************/
/**
  @brief float control_PID (float ref,float value)
            Gets control action for reaching the current
            reference.

  Example:
  @verbatim
      AC=control_PID(target_speed,speed_value);

  @endverbatim
**/

float control_PID (float ref, float value)
{
      static float q0;
      static float q1;
      static float q2;
      static float e; //current error
      static float e1; //previous error
      static float e2; //even previous error
      static float e3; //windup  error
      static float u; //current control action
      static float u1; //previous control action

      q0=0.135F;  //controler parameters init
  q1=-0.0928F;
      q2=-0.0928F;


      e=ref-value; //calculate error
      u=q0*e+q1*e3+q2*e2+u1; //calculates control action

      //varaible update

      if((u >= -100.0)&&(u <= 100.0)) //antiwindup
      {
  e3=e;
      }

  u1=u;
      e2=e1;
      e1=e;


      //control action saturation (working on +-10V here)
```

```
      if(u > 100.0F)
      {
            u=100.0F;
      }
      else if(u < -100.0F)
      {
            u=-100.0F;
      }

      return(u);
}
/*************************************************************************
**************************/
/**
  @brief float control_FT (float ref,float value)
            Gets control action for reaching the current
            reference.

  Example:
  @verbatim
      AC=control_FT(target_speed,speed_value);

  @endverbatim
**/

float control_FT (float ref, float value)
{

      static float e; //current error
      static float e1; //previous error
      static float e2; //even previous error
      static float u; //current control action
      static float u1; //previous control action
      static float u2; //previous control action


      e=ref-value; //calculate error
      u=100*e-24.583*e1-55*e2+-0.521*u1+0.2427*u2; //calculates
control action

      //varaible update

  u1=u;
      u2=u1;
      e2=e1;
      e1=e;


      //control action saturation (working on +-10V here)
      if(u > 100.0F)
      {
            u=100.0F;
      }
      else if(u < -100.0F)
      {
            u=-100.0F;
      }

      return(u);
}
```

```
/*****************************************************************
***************************/
/**
  @brief void AC2duty (float Acc_contr, float *duty1, float *duty2)
           Changes control action from +100 to -100 value
           returned by controller to duty cicle value
           required to control the motor

  Example:
  @verbatim
      AC2duty (50.0, &duty1, &duty2);
                  would return:
                  duty1=0.5
                  duty2=0
  @endverbatim
**/
void AC2duty (float Acc_contr, float *duty1, float *duty2)
{
      if (Acc_contr == 0)
      {
            *duty1=0.0F;
            *duty2=0.0F;
      }
      else if (Acc_contr > 0.0F)
      {
            *duty1=(Acc_contr/100.0F);
            *duty2=0.0F;
      }
      else if (Acc_contr < 0.0F)
      {
            *duty1=0.0F;
            *duty2=(-Acc_contr/100.0F);
      }
}
```

4.2. control.h

```
// File: control.h

#ifndef CONTROL_H
#define CONTROL_H

#include <stdint.h>
#include "stm32f4xx_hal.h"


float control_PI (float ref, float value);
float control_PD (float ref, float value);
float control_PID (float ref, float value);
float control_FT (float ref, float value);
void AC2duty (float Acc_contr, float *duty1, float *duty2);
#endif

/*** End of file
*****************************************************************
***/
```

A1.5. Código de configuración del reloj de la tarjeta e inicialización

5.1. Configuración del reloj

```
/**
  * @brief  System Clock Configuration
  *         The system Clock is configured as follow :
  *            System Clock source            = PLL (HSE)
  *            SYSCLK(Hz)                     = 180000000
  *            HCLK(Hz)                       = 180000000
  *            AHB Prescaler                  = 1
  *            APB1 Prescaler                 = 4
  *            APB2 Prescaler                 = 2
  *            HSE Frequency(Hz)              = 8000000
  *            PLL_M                          = 8
  *            PLL_N                          = 360
  *            PLL_P                          = 2
  *            PLL_Q                          = 7
  *            VDD(V)                         = 3.3
  *            Main regulator output voltage  = Scale1 mode
  *            Flash Latency(WS)              = 5
  * @param  None
  * @retval None
  */


static void SystemClock_Config(void)
{

  RCC_ClkInitTypeDef RCC_ClkInitStruct;
  RCC_OscInitTypeDef RCC_OscInitStruct;

  /* Enable Power Control clock */
  __HAL_RCC_PWR_CLK_ENABLE();

  /* The voltage scaling allows optimizing the power consumption when
      the device is clocked below the maximum system frequency, to
      update the voltage scaling value regarding system frequency
      refer to product datasheet.  */
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  /* Enable HSE Oscillator and activate PLL with HSE as source */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 360;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 7;
  if(HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    /* Initialization Error */
    Error_Handler();
  }

  if(HAL_PWREx_EnableOverDrive() != HAL_OK)
  {
    /* Initialization Error */
    Error_Handler();
```

```
  }

  /* Select PLL as system clock source and configure the HCLK, PCLK1
      and PCLK2 clocks dividers */
  RCC_ClkInitStruct.ClockType = (RCC_CLOCKTYPE_SYSCLK |
RCC_CLOCKTYPE_HCLK | RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2);
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
  if(HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) !=
HAL_OK)
  {
    /* Initialization Error */
    Error_Handler();
  }
}
static void Error_Handler(void)
{
  /* User may add here some code to deal with this error */
  while(1)
  {
            printf("Error! \n");
  }
}
```

5.2. Programa principal con inicialización. main.c

```
/**********************************************************************
* @file    main.c
  * @author  José Félix González Rojo
  * @version V1
  * @date    July-2016
  * @brief   Main program body based on CubeTemplate
**********************************************************************/
/* Includes ------------------------------------------------------------*/
#include "main.h"
#include "math.h"
#include "stdio.h"
#include "stdlib.h"
#include "stm32f4xx_hal.h"
#include "encoder.h"
#include "pwm.h"
#include "adc.h"
#include "prbs.h"

/* Private function prototypes -----------------------------------------*/

static void SystemClock_Config(void);
static void Error_Handler(void);

/**
  * @brief  Main program
  * @param  None
  * @retval None
  */

/* USER CODE BEGIN 0 */

/* Main Function */
```

```
int main(void)
{
      /* Reset of all peripherals, Initializes the Flash interface and
      the Systick. */
      HAL_Init();

      /* Configure the system clock to 180 MHz */
       SystemClock_Config();

      /* Variable Declaration */

       /* Initialize all configured peripherals */

       /* USER CODE END 2 */

       /* Infinite loop */
       /* USER CODE BEGIN WHILE */

       //while (1)
            {
            }
 }
/* USER CODE BEGIN 4 */
/* USER CODE END 4 */

/************* (C) COPYRIGHT STMicroelectronics *****END OF FILE*****/
```

A1.6. Código de configuración del contador que lleva el tiempo de muestreo

6.1. Funciones de control de HAL

```
/** @defgroup HAL_Exported_Functions_Group2 HAL Control functions
 *  @brief    HAL Control functions
 *
@verbatim

===============================================================================
                      ##### HAL Control functions #####

===============================================================================
    [..]  This section provides functions allowing to:
      (+) Provide a tick value in millisecond
      (+) Provide a blocking delay in millisecond
      (+) Suspend the time base source interrupt
      (+) Resume the time base source interrupt
      (+) Get the HAL API driver version
      (+) Get the device identifier
      (+) Get the device revision identifier
      (+) Enable/Disable Debug module during SLEEP mode
      (+) Enable/Disable Debug module during STOP mode
      (+) Enable/Disable Debug module during STANDBY mode

@endverbatim
  * @{
  */

/**
  * @brief This function is called to increment  a global variable
```

```
  *       "uwTick" used as application time base.
  * @note In the default implementation, this variable is incremented
  *         each 1ms in Systick ISR.
  * @note This function is declared as __weak to be overwritten in
  *       case of other implementations in user file.
  * @retval None
  */
__weak void HAL_IncTick(void)
{
  uwTick++;
}

/**
  * @brief Provides a tick value in millisecond.
  * @note This function is declared as __weak to be overwritten in
  *       case of other implementations in user file.
  * @retval tick value
  */
__weak uint32_t HAL_GetTick(void)
{
  return uwTick;
}

/**
  * @brief This function provides accurate delay (in milliseconds)
  *        based on variable incremented.
  * @note In the default implementation , SysTick timer is the source
  *       of time base.It is used to generate interrupts at regular
  *       time intervals where uwTick is incremented.
  * @note This function is declared as __weak to be overwritten in
  *       case of other implementations in user file.
  * @param Delay: specifies the delay time length, in milliseconds.
  * @retval None
  */
__weak void HAL_Delay(__IO uint32_t Delay)
{
  uint32_t tickstart = 0U;
  tickstart = HAL_GetTick();
  while((HAL_GetTick() - tickstart) < Delay)
  {
  }
}

/**
  * @brief Suspend Tick increment.
  * @note In the default implementation , SysTick timer is the source
  *       of time base. It is used to generate interrupts at regular
  *       time intervals. Once HAL_SuspendTick()is called, the SysTick
*         interrupt will be disabled and so Tick increment is
*         suspended.
  * @note This function is declared as __weak to be overwritten in
  *       case of other implementations in user file.
  * @retval None
  */
__weak void HAL_SuspendTick(void)
{
  /* Disable SysTick Interrupt */
  SysTick->CTRL &= ~SysTick_CTRL_TICKINT_Msk;
}
```

```
/**
  * @brief Resume Tick increment.
  * @note In the default implementation , SysTick timer is the source
  *       of time base. It is used to generate interrupts at regular
  *       time intervals. Once HAL_ResumeTick()is called, the SysTick
  *       interrupt will be enabled and so Tick increment is resumed.
  * @note This function is declared as __weak to be overwritten in
  *       case of other implementations in user file.
  * @retval None
  */
__weak void HAL_ResumeTick(void)
{
  /* Enable SysTick Interrupt */
  SysTick->CTRL  |= SysTick_CTRL_TICKINT_Msk;
}
```

6.2. Implementación para control del muestreo

Suponiendo un periodo de muestreo de 10ms y un bucle de control que itera 1000 periodos:

```
for(uint32_t i=0; i<1000;i++) //wait for sistem to stabilize
      {
            // control process start
            uint32_t Tinit = 0;
            uint32_t Tend = 0;
            Tinit = HAL_GetTick();

            // Sensor read

            //Process

            // Variable update

            // Control process end
            Tend = HAL_GetTick();

            // Wait for period end
            if (Tend-Tinit < 10)
            {
                  HAL_Delay(10-(Tend-Tinit));
            }
            else
            {
                  printf("Error! Too long procesing time \n");
                  Error_Handler();
            }
      }
```

A1.7. Código de configuración, inicialización y manejo del encoder óptico incremental

7.1. Configuración del encoder. encoder.c

```
/**
    @file encoder.c
    @brief xxx

      Required resources:
      Tim1
      PE9 (TIM1_CH1)
      PE11 (TIM1_CH2)

      Utiliced quadrature encoder Pittman.es E30 Series

    @author Angel Perles. Ed: José Félix González Rojo
    @date May 2016

*/

#include "stm32f4xx_hal.h"
void encoder_Init(void)
      {
            TIM_HandleTypeDef EncoderHandle;
            TIM_Encoder_InitTypeDef sEncoderConfig;
            GPIO_InitTypeDef  GPIO_InitStructure;

            /* PORT E ****************************************/
            __HAL_RCC_TIM1_CLK_ENABLE();
            __HAL_RCC_GPIOE_CLK_ENABLE();
            // PE.9 (TIM1_CH1) ,PE.11 (TIM1_CH2) for quadrature encoder
            GPIO_InitStructure.Pin = GPIO_PIN_9 | GPIO_PIN_11;
            GPIO_InitStructure.Pull = GPIO_PULLDOWN;
            GPIO_InitStructure.Mode = GPIO_MODE_AF_PP;
            GPIO_InitStructure.Alternate=GPIO_AF1_TIM1;
            HAL_GPIO_Init(GPIOE,&GPIO_InitStructure);

            EncoderHandle.Instance= TIM1;

            /* Prepare timer */
            __TIM1_CLK_ENABLE();

            HAL_TIM_Base_Init(&EncoderHandle);
            EncoderHandle.Init.Period = 0xffff;
            EncoderHandle.Init.Prescaler = 1;
            EncoderHandle.Init.CounterMode = TIM_COUNTERMODE_UP;

            sEncoderConfig.EncoderMode = TIM_ENCODERMODE_TI12;

            /* Use the CC1 channel to generate an interrupt in each
edge */
            sEncoderConfig.IC1Polarity = TIM_ICPOLARITY_BOTHEDGE;
            sEncoderConfig.IC1Selection = TIM_ICSELECTION_DIRECTTI;
            sEncoderConfig.IC1Prescaler = TIM_ICPSC_DIV8;
            sEncoderConfig.IC1Filter = 0x0;

            if (HAL_TIM_Encoder_Init(&EncoderHandle,&sEncoderConfig) !=
HAL_OK)
```

```
                  {
                        Error_Handler; // Error
                  }

            // Initial interuppt structures
            HAL_NVIC_ClearPendingIRQ(TIM1_CC_IRQn);
            HAL_NVIC_SetPriority(TIM1_CC_IRQn, 1, 1);
            HAL_NVIC_EnableIRQ(TIM1_CC_IRQn);

            // Count initialitation
            TIM1->CNT = 0;

            //HAL_TIM_Encoder_Start_IT(&EncoderHandle,TIM_CHANNEL_1);
            HAL_TIM_Encoder_Start(&EncoderHandle,TIM_CHANNEL_1);
      }


/**********************************************************************/
/**
  @brief Read encoder value.

  Example:
  @verbatim
      encoder_count = encoder_Read;
  @endverbatim
  */
uint32_t encoder_Read(void)
      {
            return(TIM1->CNT);
      }

/**********************************************************************/
/**
  @brief IT mode handler for interrupt detection.

  Example:
  @verbatim

  @endverbatim
  */

static void Error_Handler(void)
      {
            While(1)
            {
                  Printf("Encoder Error\n");
            }
      }

/*** End of file
**********************************************************************/
```

7.2. Configuración del encoder. encoder.h

```
// File: encoder.h

#ifndef ENCODER_H
#define ENCODER_H

#include <stdint.h>

void encoder_Init(void);
uint32_t encoder_Read(void);
static void Error_Handler(void);

#endif

/*** End of file
****************************************************************/
```

7.3. Ejemplo de uso, medida de posición y velocidad

Inicialización del encoder y toma de 1000 muestras de posición y velocidad con un periodo de muestreo de 10ms. La posición se expresa en radianes.

```
/* Main Function */
int main(void)
{
	//SystemInit() ;
	/* Reset of all peripherals, Initializes the Flash interface and
the Systick. */
	HAL_Init();

	/* Configure the system clock to 180 MHz */
	SystemClock_Config();

	/* Variable Declaration */
	float encoder_position = 0;
	float encoder_old_position = 0;
	float current_value = 0;
	float speed_value = 0;

	/* Initialize all configured peripherals */
	encoder_Init();

	for(uint32_t i=0; i<1000;i++)
	{
		// control process start
		uint32_t Tinit = 0;
		uint32_t Tend = 0;
		Tinit = HAL_GetTick();

		// Sensor read
		encoder_position = (encoder_Read()*(6.28/2000));
		speed_value = abs(encoder_position -
encoder_old_position)/0.01;

		// Variable update
```

```
            encoder_old_position = encoder_position;

            // Control process end
            Tend = HAL_GetTick();
            // Wait for period end
            if (Tend-Tinit < 10)
            {
                  HAL_Delay(10-(Tend-Tinit));
            }
            else
            {
                  printf("Error! Too long procesing time \n");
                  Error_Handler();
            }
      }
}
```

A1.8. Código de configuración, inicialización y manejo del ADC

8.1. Configuración del ADC. adc.c

```
/**
    @file adc.c
    @brief xxx

      Required resources:
            TIM2
            PA1 (TIM2_CH2,ADC123_IN)

      Using adc input with l298 full H-bridge current sensing output
      and a signal condition circuit to control a dc pittman motor
      with torque control.

      Now we will try to raise the ADC performance.
      Replace ADC_SAMPLETIME_480CYCLES with ADC_SAMPLETIME_28CYCLES.
      You will notice that the loop in main() is never executed.
      You can use the debugger to find out that by the time the ADC
      value is read, the next ADC interrupt is already pending.

      This happens because the ADC reads new values faster than our
      code can handle. To speed this up, we will use the DMA to get
      the ADC automatically write values into a buffer in RAM and only
      call the CPU when the entire buffer is written. According to the
      STM32F4 reference manual, ADC1 is connected to channel 0 of
      streams 0 and 4 of DMA2.

      To use DMA we have to configure it.

       (##) In case of using DMA to control data transfer (e.g.
      HAL_ADC_Start_DMA())
            (+++) Enable the DMAx interface clock using
      __HAL_RCC_DMAx_CLK_ENABLE()
            (+++) Configure and enable two DMA streams (just one for input
      only) stream for managing data transfer from peripheral to memory
      (output stream)
            (+++) Associate the initialized DMA handle to the CRYP DMA handle
      using  __HAL_LINKDMA()
            (+++) Configure the priority and enable the NVIC for the transfer
      complete interrupt on the two DMA Streams. The output stream should have
      higher priority than the input stream.
```

```
            (#) Configure the ADC parameters (resolution, data alignment,...)
       and regular group parameters (conversion trigger, sequencer, ...)
       using function HAL_ADC_Init().

            (#) Configure the channels for regular group parameters (channel
       number, channel rank into sequencer, ..., into regular group)using
       function HAL_ADC_ConfigChannel().

      *** DMA mode IO operation ***
      ==============================
      [..]
        (+) Start the ADC peripheral using HAL_ADC_Start_DMA(), at this
       stage the user specify the length of data to be transferred at
       each end of conversion
        (+) At The end of data transfer by HAL_ADC_ConvCpltCallback()
       function is executed and user can add his own code by
       customization of function pointer HAL_ADC_ConvCpltCallback
        (+) In case of transfer Error, HAL_ADC_ErrorCallback() function
       is executed and user can add his own code by customization of
       function pointer HAL_ADC_ErrorCallback
        (+) Stop the ADC peripheral using HAL_ADC_Stop_DMA()

    @author Jose Felix González Rojo
    @date May 2016
*/

#include "stm32f4xx_hal.h"
#include "adc.h"

uint32_t ADCValue; //Converted value
uint32_t MeasurementNumber;
enum{ ADC_BUFFER_LENGTH = 8192 };
uint32_t ADCBuffer[ADC_BUFFER_LENGTH];

int DmaOffsetBeforeAveragingF, DmaOffsetAfterAveragingF;
int DmaOffsetBeforeAveragingH, DmaOffsetAfterAveragingH;

/* Handlers for both functions */
DMA_HandleTypeDef  DmaHandle;
ADC_HandleTypeDef AdcHandle;

/* ADC init function */

void Adc_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
        ADC_ChannelConfTypeDef adcChannel;
        TIM_HandleTypeDef TimHandle;

            // GPIO setup
    __GPIOA_CLK_ENABLE();

    GPIO_InitStructure.Pin = GPIO_PIN_1;
    GPIO_InitStructure.Mode = GPIO_MODE_ANALOG;
    GPIO_InitStructure.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStructure);

            // ADC setup
    __ADC1_CLK_ENABLE();
    AdcHandle.Instance = ADC1;
```

```
    AdcHandle.Init.ClockPrescaler = ADC_CLOCKPRESCALER_PCLK_DIV2;
    AdcHandle.Init.Resolution = ADC_RESOLUTION_12B;
            AdcHandle.Init.DataAlign = ADC_DATAALIGN_RIGHT;
    AdcHandle.Init.ScanConvMode = DISABLE;
            //AdcHandle.Init.ScanConvMode = ENABLE; //for a second
channel
            AdcHandle.Init.EOCSelection = DISABLE;
    AdcHandle.Init.ContinuousConvMode = ENABLE;
    AdcHandle.Init.NbrOfConversion = 1;
            //AdcHandle.Init.NbrOfConversion = 2; //for a second
channel
    AdcHandle.Init.DiscontinuousConvMode = DISABLE;
    AdcHandle.Init.NbrOfDiscConversion = 0;
    AdcHandle.Init.ExternalTrigConv = ADC_EXTERNALTRIGCONV_T1_CC1;
            //AdcHandle.Init.ExternalTrigConv =
ADC_EXTERNALTRIGCONV_T2_CC2;
    AdcHandle.Init.ExternalTrigConvEdge =
ADC_EXTERNALTRIGCONVEDGE_NONE;
    AdcHandle.Init.DMAContinuousRequests = ENABLE;

            if (HAL_ADC_Init(&AdcHandle) != HAL_OK)
    {
        Error_Handler();
    }

            // Channel setup
    adcChannel.Channel = ADC_CHANNEL_1;
    adcChannel.Rank = 1;
    //adcChannel.SamplingTime = ADC_SAMPLETIME_480CYCLES;
            adcChannel.SamplingTime = ADC_SAMPLETIME_3CYCLES; //3CYCLES
is the maximun speed I don't think less than 28 is worth but...
    adcChannel.Offset = 0;

            if (HAL_ADC_ConfigChannel(&AdcHandle, &adcChannel) !=
HAL_OK)
    {
        Error_Handler();
    }

            /* //for a second channel with priority 2
    adcChannel.Channel = ADC_CHANNEL_16;
    adcChannel.Rank = 2;
    adcChannel.SamplingTime = ADC_SAMPLETIME_480CYCLES;
    adcChannel.Offset = 0;

    if (HAL_ADC_ConfigChannel(&AdcHandle, &adcChannel) != HAL_OK)
    {
        Error_Handler();
    }*/

            /* DMA setup */
    __DMA2_CLK_ENABLE();
    DmaHandle.Instance = DMA2_Stream4;

    DmaHandle.Init.Channel  = DMA_CHANNEL_0;
    DmaHandle.Init.Direction = DMA_PERIPH_TO_MEMORY;
    DmaHandle.Init.PeriphInc = DMA_PINC_DISABLE;
    DmaHandle.Init.MemInc = DMA_MINC_ENABLE;
    DmaHandle.Init.PeriphDataAlignment = DMA_PDATAALIGN_WORD;
    DmaHandle.Init.MemDataAlignment = DMA_MDATAALIGN_WORD;
```

```
    DmaHandle.Init.Mode = DMA_CIRCULAR;
    DmaHandle.Init.Priority = DMA_PRIORITY_HIGH;
    DmaHandle.Init.FIFOMode = DMA_FIFOMODE_DISABLE;
    DmaHandle.Init.FIFOThreshold = DMA_FIFO_THRESHOLD_HALFFULL;
    DmaHandle.Init.MemBurst = DMA_MBURST_SINGLE;
    DmaHandle.Init.PeriphBurst = DMA_PBURST_SINGLE;

            if (HAL_DMA_Init(&DmaHandle) != HAL_OK)
    {
        Error_Handler();
    }

    __HAL_LINKDMA(&AdcHandle, DMA_Handle, DmaHandle);

            //IT setup
            /*HAL_NVIC_SetPriority(ADC_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(ADC_IRQn);*/
    HAL_NVIC_SetPriority(DMA2_Stream4_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(DMA2_Stream4_IRQn);

            // Starts ADC
            //HAL_ADC_Start(&AdcHandle); //Polling mode
            //HAL_ADC_Start_IT(&AdcHandle); //IT mode
            HAL_ADC_Start_DMA(&AdcHandle, ADCBuffer,
ADC_BUFFER_LENGTH); //DMA mode
}

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* AdcHandle) //for
precise data
{
      DmaOffsetBeforeAveragingF = ADC_BUFFER_LENGTH - DMA2_Stream4-
>NDTR;
      ADCValue=0; // getting average value from buffer
      for(uint32_t i=ADC_BUFFER_LENGTH/2; i<ADC_BUFFER_LENGTH; i++)
      {
            ADCValue = ADCValue + ADCBuffer[i];
      }
      ADCValue = ADCValue / (ADC_BUFFER_LENGTH/2);
      DmaOffsetAfterAveragingF = ADC_BUFFER_LENGTH - DMA2_Stream4-
>NDTR;
      MeasurementNumber += ADC_BUFFER_LENGTH/2;
}

void HAL_ADC_ConvHalfCpltCallback(ADC_HandleTypeDef* AdcHandle) //for
precise data using half of a buffer with each function
{
      DmaOffsetBeforeAveragingH = ADC_BUFFER_LENGTH - DMA2_Stream4-
>NDTR;
      ADCValue=0; // getting average value from buffer
      for(uint32_t i=0; i<ADC_BUFFER_LENGTH/2; i++)
      {
            ADCValue = ADCValue + ADCBuffer[i];
      }
      ADCValue = ADCValue / (ADC_BUFFER_LENGTH/2);
      DmaOffsetAfterAveragingH = ADC_BUFFER_LENGTH - DMA2_Stream4-
>NDTR;
      MeasurementNumber += ADC_BUFFER_LENGTH/2;
}

void DMA2_Stream4_IRQHandler(void)
```

```
{
      HAL_DMA_IRQHandler(&DmaHandle);
}

void ADC_IRQHandler(void)
{
  HAL_ADC_IRQHandler(&AdcHandle);
}

//=============================================================//
/*ADC Read funciton*/
uint32_t ADC_Read(void)
{
      return(ADCValue);
}

//=============================================================//
/*Error Handler function*/
static void Error_Handler(void)
{
  /* User may add here some code to deal with this error */
  while(1)
  {
            printf("Error! \n");
  }
}

/*** End of file
*****************************************************************/
```

8.2. Configuración del ADC. adc.h

```
// File: adc.h
#ifndef ADC_H
#define ADC_H

#include <stdint.h>
#include "stm32f4xx_hal.h"

void Adc_Init(void);
void Dma_Init(void);
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* AdcHandle);
void HAL_ADC_ConvHalfCpltCallback(ADC_HandleTypeDef* AdcHandle);
void DMA2_Stream4_IRQHandler(void);
void ADC_IRQHandler(void);
uint32_t ADC_Read(void);
static void Error_Handler(void);

#endif

/*** End of file
*****************************************************************/
```

8.3. Ejemplo de uso midiendo la corriente del motor

Inicialización del ADC y toma de 1000 muestras de la corriente del motor con un periodo de muestreo de 10ms. La corriente se expresa sin coma flotante y como una medida de tensión. Para obtener el valor real, hay que dividir entre 0.5 que es el valor de

la resistencia de medida y dividir entre 1000 para obtener la posición de la coma y expresar el resultado en A.

```
/* Includes ------------------------------------------------------*/
#include "main.h"
#include "math.h"
#include "stdio.h"
#include "stdlib.h"
#include "stm32f4xx_hal.h"
#include "adc.h"


/* Private typedef -----------------------------------------------*/
/* Private define ------------------------------------------------*/
/* Private macro -------------------------------------------------*/
/* Private variables ---------------------------------------------*/
/* Private function prototypes -----------------------------------*/

static void SystemClock_Config(void);
static void Error_Handler(void);

/* Private function prototypes -----------------------------------*/
/* Private functions ---------------------------------------------*/

/**
  * @brief  Main program
  * @param  None
  * @retval None
*/

/* USER CODE BEGIN 0 */

/* Main Function */
int main(void)
{
      //SystemInit() ;
      /* Reset of all peripherals, Initializes the Flash interface and
the Systick. */
      HAL_Init();

       /* Configure the system clock to 180 MHz */
      SystemClock_Config();

      float current_value = 0;

      /* Initialize all configured peripherals */
      Adc_Init();

      /* USER CODE END 2 */

      /* Infinite loop */
      /* USER CODE BEGIN WHILE */

      for(uint32_t i=0; i<1000;i++)
      {
            // control process start
            uint32_t Tinit = 0;
            uint32_t Tend = 0;
            Tinit = HAL_GetTick();
```

```
            // Sensor read
            current_value = ADC_Read();

            //Process

            // Variable update

            // Control process end
            Tend = HAL_GetTick();
            // Wait for period end
            if (Tend-Tinit < 10)
            {
                  HAL_Delay(10-(Tend-Tinit));
            }
            else
            {
                  printf("Error! Too long procesing time \n");
                  Error_Handler();
            }
      }
}
/* USER CODE BEGIN 4 */
/* USER CODE END 4 */

/************* (C) COPYRIGHT STMicroelectronics *****END OF FILE*****/
```

A1.9. Código de configuración, inicialización y manejo del generador de señal PWM

9.1. Configuración del generador de señal PWM. pwm.c

```
/**
    @file pwm.c
    @brief xxx

      Required resources:
      Tim9
      PC6 (TIM8_CH1)
      PC7 (TIM8_CH2)

      Using pwm on l298 full H-bridge to control a dc pittman motor

        TIM8 is connected to APB2 bus, which has on F429 device
            which works at TIMxCLK MHz by default, and internal PLL
increase
    this to up to 180MHz

    Set timer prescaller
    Timer count frequency is set with

    timer_tick_frequency = Timer_default_frequency / (prescaller_set +
1)

    In our case, we want a max frequency for timer, so we set
prescaller to 0
    And our timer will have tick frequency

    timer_tick_frequency = 180000000 / (0 + 1) = 180000000
```

```
                Set timer period when it have reset
    First you have to know max value for timer
    In our case it is 16bit = 65535
    To get your frequency for PWM, equation is simple

    PWM_frequency = timer_tick_frequency / (TIM_Period + 1)

    If you know your PWM frequency you want to have timer period set
correct

    TIM_Period = timer_tick_frequency / PWM_frequency - 1

    In our case, for 16Khz PWM_frequency, set Period to

    TIM_Period = 180000000 / 16000 - 1 = 11250

    If you get TIM_Period larger than max timer value (in our case
65535),
    you have to choose larger prescaler and slow down timer tick
frequency

    @author Jose Felix Gonzalez
    @date May 2016
*/

#include "stm32f4xx_hal.h"
#include "pwm.h"

TIM_HandleTypeDef TimHandle;
/* TIM8 init function */
void Pwm_Init(void)
{
        GPIO_InitTypeDef GPIO_InitStructure;
        //TIM_HandleTypeDef TimHandle;
        TIM_OC_InitTypeDef PWMConfig;
        TIM_BreakDeadTimeConfigTypeDef DeadConfig;

                        // GPIO setup
        __GPIOC_CLK_ENABLE();

        GPIO_InitStructure.Pin = GPIO_PIN_6 | GPIO_PIN_7;
        GPIO_InitStructure.Mode = GPIO_MODE_AF_PP;
        GPIO_InitStructure.Pull = GPIO_PULLUP;
        GPIO_InitStructure.Speed = GPIO_SPEED_HIGH;
        GPIO_InitStructure.Alternate = GPIO_AF3_TIM8;
        HAL_GPIO_Init(GPIOC, &GPIO_InitStructure);

                        // Timer setup
        __TIM8_CLK_ENABLE();

        //uint16_t uwPrescalerValue =0;
        TimHandle.Instance = TIM8;
        TimHandle.Init.Period = 11250; //max for 16 b is 65535
        TimHandle.Init.Prescaler = 0; //uwPrescalerValue;
        TimHandle.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
        TimHandle.Init.CounterMode = TIM_COUNTERMODE_UP;

        if(HAL_TIM_PWM_Init(&TimHandle) != HAL_OK)
                        {
```

```
                              printf("Error: HAL_TIM_PWM_Init \n");
                              Error_Handler();
                        }

                        // Pwm setup
        PWMConfig.Pulse = 0; //65535;
        PWMConfig.OCMode = TIM_OCMODE_PWM1;
        PWMConfig.OCPolarity = TIM_OCPOLARITY_HIGH;
        PWMConfig.OCNPolarity = TIM_OCNPOLARITY_HIGH;
        PWMConfig.OCIdleState = TIM_OCIDLESTATE_SET;
        PWMConfig.OCNIdleState= TIM_OCNIDLESTATE_RESET;
        PWMConfig.OCFastMode = TIM_OCFAST_DISABLE;
//TIM_OCFAST_ENABLE;


if(HAL_TIM_PWM_ConfigChannel(&TimHandle,&PWMConfig,TIM_CHANNEL_1) !=
HAL_OK)
                        {
                              printf("Error: HAL_TIM_PWM_ConfigChannel
\n");
          Error_Handler();
                        }

      if(HAL_TIM_PWM_ConfigChannel(&TimHandle,&PWMConfig,TIM_CHANNEL_2
) != HAL_OK)
                        {
                              printf("Error: HAL_TIM_PWM_ConfigChannel
\n");
          Error_Handler();
                        }

        DeadConfig.AutomaticOutput=TIM_AUTOMATICOUTPUT_ENABLE;
        DeadConfig.BreakPolarity = TIM_BREAKPOLARITY_LOW;
        DeadConfig.BreakState = TIM_BREAK_DISABLE;
        DeadConfig.DeadTime = 0xFF;
        DeadConfig.LockLevel = TIM_LOCKLEVEL_OFF;
        DeadConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
        DeadConfig.OffStateRunMode = TIM_OSSR_DISABLE;

        if(HAL_TIMEx_ConfigBreakDeadTime(&TimHandle,&DeadConfig) !=
HAL_OK)
            {
            printf("Error: HAL_TIMEx_ConfigBreakDeadTime \n");
            Error_Handler();
            }

        if(HAL_TIM_PWM_Start(&TimHandle, TIM_CHANNEL_1) != HAL_OK)
            {
            printf("Error: HAL_TIM_PWM_Start \n");
            Error_Handler();
            }
        if(HAL_TIM_PWM_Start(&TimHandle, TIM_CHANNEL_2) != HAL_OK)
            {
            printf("Error: HAL_TIM_PWM_Start \n");
            Error_Handler();
            }
        if(HAL_TIMEx_PWMN_Start(&TimHandle,TIM_CHANNEL_1) != HAL_OK)
            {
            printf("Error: HAL_TIMEx_PWMN_Start \n");
            Error_Handler();
            }
```

```
      if(HAL_TIMEx_PWMN_Start(&TimHandle,TIM_CHANNEL_2) != HAL_OK)
            {
            printf("Error: HAL_TIMEx_PWMN_Start \n");
            Error_Handler();
            }
}
/*****************************************************************/

void pwm_Set(uint32_t channel, double duty)
      {
            if (channel == TIM_CHANNEL_1)
            {
                  TIM8->CCR1=(11250+1)*duty;
            }
            else if (channel == TIM_CHANNEL_2)
            {
                  TIM8->CCR2=(11250+1)*duty;
            }
      }
/*****************************************************************/

void pwm_Stop(uint32_t channel)
{
      HAL_TIM_PWM_Stop(&TimHandle, channel);
}
/* PWM set to a value */


/*****************************************************************/

static void Error_Handler(void)
{
  /* User may add here some code to deal with this error */
  while(1)
  {
            printf("Error! \n");
  }
}


/*** End of file
*****************************************************************/
```

9.2. Configuración del generador de señal PWM. pwm.h

```
// File: pwm.h

#ifndef PWM_H
#define PWM_H

#include <stdint.h>
#include "stm32f4xx_hal.h"

#define PERIOD 11250
/*TIM_Period = 180000000 / DESIREDFRECUENCI - 1 for 16kHz it is
TIM_Period = 180000000 / 16000 - 1 = 11250*/
```

```
void Pwm_Init(void);
void pwm_Set(uint32_t channel, float duty);
void pwm_Stop(uint32_t channel);
static void Error_Handler(void);

#endif

/*** End of file
****************************************************************/
```

9.3. Ejemplo de uso generando una señal en dos canales

El ejemplo siguiente genera de forma alternativa una señal PWM en dos pines, cambiando el ciclo de trabajo de la misma.

```
/* Includes ------------------------------------------------------
--------*/
#include "main.h"
#include "math.h"
#include "stdio.h"
#include "stdlib.h"
#include "stm32f4xx_hal.h""
#include "pwm.h"


/* Private typedef -----------------------------------------------*/
/* Private define ------------------------------------------------*/
/* Private macro -------------------------------------------------*/
/* Private variables ---------------------------------------------*/
/* Private function prototypes -----------------------------------*/
static void SystemClock_Config(void);
static void Error_Handler(void);
/* Private function prototypes -----------------------------------*/
/* Private functions ---------------------------------------------*/
/* USER CODE BEGIN 0 */

/* Main Function */
int main(void)
{
      //SystemInit() ;
      /* Reset of all peripherals, Initializes the Flash interface and
the Systick. */
       HAL_Init();

  /* Configure the system clock to 180 MHz */
       SystemClock_Config();


  /* Initialize all configured peripherals */
      Pwm_Init();

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
      While(1)
      {
```

```
            pwm_Set(TIM_CHANNEL_1, 0);
            pwm_Set(TIM_CHANNEL_2, 0);
            HAL_Delay(5000);
            pwm_Set(TIM_CHANNEL_1, 1);
            pwm_Set(TIM_CHANNEL_2, 0);
            HAL_Delay(5000);
            pwm_Set(TIM_CHANNEL_1, 0.75);
            pwm_Set(TIM_CHANNEL_2, 0);
            HAL_Delay(5000);
            pwm_Set(TIM_CHANNEL_1, 0.5);
            pwm_Set(TIM_CHANNEL_2, 0);
            HAL_Delay(5000);
            pwm_Set(TIM_CHANNEL_1, 0.25);
            pwm_Set(TIM_CHANNEL_2, 0);
            HAL_Delay(5000);
            pwm_Set(TIM_CHANNEL_1, 0);
            pwm_Set(TIM_CHANNEL_2, 1);
            HAL_Delay(5000);
            pwm_Set(TIM_CHANNEL_1, 0);
            pwm_Set(TIM_CHANNEL_2, 0.75);
            HAL_Delay(5000);
            pwm_Set(TIM_CHANNEL_1, 0);
            pwm_Set(TIM_CHANNEL_2, 0.5);
            HAL_Delay(5000);
            pwm_Set(TIM_CHANNEL_1, 0);
            pwm_Set(TIM_CHANNEL_2, 0.25);
            HAL_Delay(5000);
            pwm_Set(TIM_CHANNEL_1, 1);
            pwm_Set(TIM_CHANNEL_2, 1);
            HAL_Delay(5000);


      }

      pwm_Stop(TIM_CHANNEL_1);
      pwm_Stop(TIM_CHANNEL_2);
}
/************** (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

A1.10. Código de generación de una señal PRBS para PWM

10.1. prbs.c

```
/**
    @file prbs.c
    @brief xxx

      Required resources:
            none

     *** PRBS value generation changing PWM sig ***
     ===============================================


    @author Jose Felix Gonzalez
    @date May 2016
```

```
*/

#include "stm32f4xx_hal.h"
#include <stdio.h>
#include <stdlib.h>
#include "prbs.h"

//=========================================
int8_t Generador;

/* duty for PWM from PRBS */
/*****************************************************************
**************************/
/**
  @brief Returns duty cicle to produce a PWM
      output signal based on a PRBS generated funtion.
      If input max volt for pwm is 10v, 5V average means 0.5 duty
      and 1V average means 0.1 duty.
      Deviation is a percent afecting duty that can be positive or
negative
      it is recomended to use 0.1
      Elong is not used for this purpose but included for future
improvements

  Example:
  @verbatim
      duty =PWM_duty_prbs(5,0.1,5);
  @endverbatim
  */
float PWM_duty_prbs(float Average,float deviation,int Elong)
{
      float duty;

            if(rand()%2 == 1)
            {
                  Generador = 1;
                  //printf("OUTPUT HIGH\n",Generador[i]);
            }
            else
            {
                  Generador = -1;
            }

        duty = Generador*deviation+(Average*0.1F);

      return (duty);
}

/*** End of file
*****************************************************************/
```

10.2. prbs.h

```
// File: prbs.h

#ifndef PRBS_H
#define PRBS_H

#include <stdint.h>
#include "stm32f4xx_hal.h"

float PWM_duty_prbs(float Average,float deviation,int Elong);

#endif

/*** End of file
****************************************************************/
```

A1.11. Código de generación de referencias mediante perfiles de movimiento

11.1. refgentray.c

```
/**
    @file refgentray.c
    @brief xxx

            Initialices a trayectory between 2 position points at
            given max. speed and aceleration. Also returns every
            speed reference value untill trayectory is completed

    @author Jose Felix
    @date May 2016

*/

//Uncomment the next line to use debugg printf on aplication
//#define DEBUGGING_OUTPUT

#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <math.h>
#include "refgentray.h"

 float spd_inc=0;
 float pos_inc=0;
 float dt=0;
 float a=0;
 float a_t_discrete=0;
 float cte_spd_pos_discrete=0;
 uint8_t skip_sspeed=0;
 int8_t sign=0;
volatile uint32_t iterations=0;
 float pos_i=0;
 float pos_f=0;
 float spd_i=0;
 float spd_f=0;
```

```
 float aspl=0;
 float b=0;
 float c=0;
 float d=0;
/*****************************************************************/
/**
  @brief uint32_t trapez_motion_Init(float pos_i, float pos_f, float
spd_i, float spd_max, float a)
      Initialices and calcules the main parameters of a trayectory
between pos_i and pos_f,
      returning the amount of iterations it takes to reach it at given
max speed and aceletation.

  Example:
  @verbatim
      it=trapez_motion_Init(0, 300, 0, 15, 5);

  @endverbatim
**/
uint32_t trapez_motion_Init(float pos_ini, float pos_fin, float
spd_ini, float spd_max, float acel, uint32_t *iterations)
{
//Based on Misan's DC servomotor Now in a function
//Limitation(s): assumes 0 initial speed (spd_i is useless, should be
0)
            static float d_pos=0;
            static float d_spd=0;
            static float a_t=0;
            static float acc_pos=0;
            static float cte_spd_pos=0;

            pos_i=pos_ini;
            pos_f=pos_fin;
            spd_i=spd_ini;
            a=acel;

    dt = TRAPEZ_DT;  //10ms tiempo entre iteraciones
    skip_sspeed = 0;

        if(pos_f > pos_i)
            {
      sign=1;
    }
            else
    {
                  sign=-1;
    }

    d_pos = pos_f - pos_i;    //Difference in position
    d_spd = spd_max - spd_i;        //Difference in speed
    a_t = d_spd / a;          //How long do we accelerate?
    a_t_discrete = a_t / dt;        //(in ticks)
    spd_inc = d_spd / a_t_discrete; //Every tick, increase spd by
    //Position from acc:
    for (uint32_t i=0;i<a_t_discrete;i++)
            {
        acc_pos = acc_pos + (i*spd_inc*dt);
            }
```

```
    //It's possible to overshoot position if the acceleration is too
low.
    //In that case we should sacrifice the top speed
    if((2*acc_pos) > sign*d_pos)
            {
        #ifdef DEBUGGING_OUTPUT
        printf("Position overshoot.\n");
        #endif

        spd_max = sqrt(a*sign*d_pos);

        //Redo the initial math:
        d_spd = spd_max - spd_i;        //Difference in speed
        a_t = d_spd / a;                //How long do we accelerate?
        a_t_discrete = a_t / dt;        //(in ticks)
        spd_inc = d_spd / a_t_discrete; //Every tick, increase spd by
        //Position from acc:
        acc_pos = 0;
        for (uint32_t i=0;i<a_t_discrete;i++)
                        {
            acc_pos = acc_pos + (i*spd_inc*dt);
        }
                  }

    cte_spd_pos = sign*d_pos - 2*acc_pos; //calculate the position
where speed will be constant
    cte_spd_pos_discrete = (cte_spd_pos/spd_max) / dt; //how long will
speed be steady?
    if(cte_spd_pos_discrete < 0)
            {
                    #ifdef DEBUGGING_OUTPUT
        printf("No steady speed!.\n");
        #endif
        skip_sspeed = 1; //we skip steady speed if negative
                        cte_spd_pos_discrete=0;
    }

    *iterations = 2*a_t_discrete + cte_spd_pos_discrete;
            if (iterations !=0)
            {
                  return(1);
            }
            else
            {
                  return(0);
            }
}
/*****************************************************************/
/**
  @brief uint32_t ramp_motion_Init(float pos_i, float pos_f, float
spd_i, float spd_max, float a)
      Initialices and calcules the main parameters of a trayectory
between pos_i and pos_f,
      returning the amount of iterations it takes to reach it at given
max speed and aceletation.

  Example:
  @verbatim
      it=ramp_motion_Init(0, 300, 0, 15, 5);
```

```
  @endverbatim
**/
uint32_t ramp_motion_Init(float pos_ini, float pos_fin, float spd_ini,
float spd_max, float acel, uint32_t *iterations)
{
      float d_pos=0;
      float d_spd=0;
      static int8_t sign;
      pos_i=pos_ini;
      pos_f=pos_fin;
      spd_i=spd_ini;
      spd_f=spd_max;
    dt = TRAPEZ_DT;  //10ms tiempo entre iteraciones

    if(pos_f > pos_i)
      {
      sign=1;
      }
    else
    {
      sign=-1;
    }

    d_pos = pos_f - pos_i;    //Difference in position
    d_spd = spd_max - spd_i;        //Difference in speed
    pos_inc = sign*d_spd*dt;

    *iterations=d_pos/pos_inc;

      if (iterations !=0)
      {
            return(1);
      }
      else
      {
            return(0);
      }
}

/*****************************************************************/
/**
  @brief uint32_t spline_motion_Init(float pos_i, float pos_f, float
spd_i, float spd_max, float a)
      Initialices and calcules the main parameters of a trayectory
between pos_i and pos_f,
      returning the amount of iterations it takes to reach it at given
max speed and aceletation.

  Example:
  @verbatim
      it=spline_motion_Init(0, 300, 0, 15, 5);

  @endverbatim
**/
uint32_t spline_motion_Init(float pos_ini, float pos_fin, float
spd_ini, float spd_max, float acel, uint32_t *iterations)
{
            dt = TRAPEZ_DT;

             aspl= pos_ini;
             b= spd_ini;
```

```
            c=3*( pos_fin - pos_ini)-2*spd_ini+ spd_max;
            d=2*( pos_ini - pos_fin)+ spd_ini + spd_max;

           *iterations =(int)(TFIN/dt);

      if (iterations !=0)
      {
            return(1);
      }
      else
      {
            return(0);
      }
}


/*****************************************************************/

/**
  @brief float trapez_motion_ref (uint32_t *i)
           returns the speed reference to calculate
           the control action to make the moovement each iteration.

  Example:
  @verbatim
      speed_ref = trapez_motion_ref (&i)

  @endverbatim
**/
float trapez_motion_ref (uint32_t i)
{
      static float old_spd;
      static float ref_spd;
      static float ref_pos;
      static float ref_acc;
      static float sum_spd;

      if(i<iterations)
      {
    if (i ==0)
            {
                  ref_spd=spd_i;
                  ref_pos=pos_i;
                  ref_acc=sign*a;
                  old_spd=ref_spd;
                  sum_spd =0;
                  return(ref_spd);
    }
    if ((i >0)&&(i < (uint32_t)a_t_discrete))
            {
        ref_spd = old_spd + sign*spd_inc;
                        sum_spd = sum_spd + old_spd;
        ref_pos = pos_i + sum_spd*dt;
        ref_acc = sign*a;
        old_spd = ref_spd;
                        return(ref_spd);
    }
    if(skip_sspeed == 0)
            {
        //Constant speed
```

```
        if ((i >= (uint32_t)a_t_discrete)&&(i <
(uint32_t)(a_t_discrete + cte_spd_pos_discrete)))
                        {
            ref_spd = old_spd;
            ref_pos = pos_i + sum_spd*dt;
            ref_acc = 0;
            old_spd = ref_spd;
            sum_spd = sum_spd + old_spd;
                                return(ref_spd);
                        }
    }
    //Negative Acceleration:
    if (i >= (uint32_t)(a_t_discrete + cte_spd_pos_discrete))
            {
        ref_spd = old_spd - sign*spd_inc;
        ref_pos = pos_i + sum_spd*dt;
        ref_acc = -(sign*a);
        old_spd = ref_spd;
        sum_spd = sum_spd + old_spd;
                        return(ref_spd);
            }
      }
}
/*****************************************************************/
/**
  @brief float ramp_motion_ref (uint32_t *i)
            returns the speed reference to calculate
            the control action to make the moovement each iteration.

  Example:
  @verbatim
      speed_ref = ramp_motion_ref (&i)

  @endverbatim
**/
float ramp_motion_ref (uint32_t i)
{
      static float old_spd;
      static float ref_spd;
      static float ref_pos;
      static float ref_acc;
      static float sum_pos;

      if(i<iterations)
      {
            ref_spd = sign*spd_f;
            ref_pos = pos_i + pos_inc*i;
            ref_acc = sign*a;
            return(ref_pos);
      }
}
/*****************************************************************/
/**
  @brief float spline_motion_ref (uint32_t *i)
            returns the speed reference to calculate
            the control action to make the moovement each iteration.

  Example:
  @verbatim
      speed_ref = spline_motion_ref (&i)
```

```
  @endverbatim
**/
float spline_motion_ref (uint32_t i)
{
      float ref_pos;
      float t;
      if(i<iterations)
      {
      t=i*SPL_INC;
      ref_pos=a+b*t+c*t*t+d*t*t*t;
      return(ref_pos);
      }
}
```

11.2. refgentray.h

```
// File: refgentray.h

#ifndef REFGENTRAY_H
#define REFGENTRAY_H

#include <stdint.h>
#include "stm32f4xx_hal.h"

#define TRAPEZ_DT 0.01  //Trapezoidal timebase. Has to match hardware!
#define SPL_INC 0.001   //Spline increment.Care on not tested val.
#define TFIN 8 //generation end time

uint32_t trapez_motion_Init(float pos_ini, float pos_fin, float
spd_ini, float spd_max, float acel, uint32_t *iterations);
uint32_t ramp_motion_Init(float pos_ini, float pos_fin, float spd_ini,
float spd_max, float acel, uint32_t *iterations);
uint32_t spline_motion_Init(float pos_ini, float pos_fin, float
spd_ini, float spd_max, float acel, uint32_t *iterations);
float trapez_motion_ref (uint32_t i);
float ramp_motion_ref (uint32_t i);
float spline_motion_ref (uint32_t i);

#endif

/*** End of file
****************************************************************/
```

11.3. Script de función de Matlab para generar una trayectoria de tercer orden para simulaciones

```
function [jerk, spd, pos, acc ] = third_motion_2( pos_i, pos_f, spd_i,
spd_max, a,j )
%Based on trapez motion now getting 3rd order S profile
%Limitation(s): assumes 0 initial speed (spd_i is useless, should be
0) also needs extremly high jerk. Tested values are:
    %pos_i=0
    %pos_f=3750
    %spd_i=0
    %spd_max=800
    %a=1600
    %j=3200

    dt = 0.01;  %10ms
    skip_sacel = 0;
```

```
    if(pos_f > pos_i)
        sign=1
    else
        sign=-1
    end
    d_pos = pos_f - pos_i;          %Difference in position
    d_spd = spd_max - spd_i;        %Difference in speed
    d_accel=a;
    j_t=d_accel/j;
    j_t_discrete = j_t/dt;
    acc_inc = d_accel/j_t_discrete
    a_t = d_spd / a;                %How long do we accelerate?
    a_t_discrete = a_t / dt        %    (in ticks)
    spd_inc = d_spd / a_t_discrete %Every tick, increase spd by
    %Position from acc:
    j_pos = 0;
    for i=1:j_t_discrete
        j_pos=j_pos+(i*acc_inc*dt);
    end
    if((2*j_pos) > sign*d_spd)
        disp('Position overshoot')

        a = sqrt(j*sign*d_spd)

        %Redo the initial math:
        d_accel = a;        %Difference in speed
        j_t=d_accel/j;
        j_t_discrete = j_t/dt;
        acc_inc = d_accel / j_t_discrete %Every tick, increase spd by
        %Position from acc:
        j_pos = 0;
        for i = 1:j_t_discrete
            j_pos = j_pos + (i*acc_inc*dt);
        end
        j_pos
    end

    cte_acc_pos = d_spd - 2*j_pos;
    cte_acc_pos_discrete = (cte_acc_pos/a) / dt
    if(cte_acc_pos_discrete < 0)
        disp('No steady acc!')
        skip_sacel = 1;
        cte_acc_pos_discrete=0;
        %cte_acc_pos=0;
    end



    if(skip_sacel ==1)
    cte_spd_pos = sign*d_pos - 2*(2*j_pos);
    cte_spd_pos_discrete = (cte_spd_pos/spd_max) / dt
    else
    cte_spd_pos = sign*d_pos - 2*(2*j_pos+cte_acc_pos_discrete);
    cte_spd_pos_discrete = (cte_spd_pos/spd_max) / dt
    end
    %At this point all the parameters are computed, we can get the 3
plots
    %vector_length = 2*length(a_t_discrete) +
length(cte_spd_pos_discrete);
    spd = zeros(1,1);
    %spd = zeros(1,vector_length);
```

```
spd(1) = spd_i;
pos = zeros(1,1);
%pos = zeros(1,vector_length);
pos(1) = pos_i;
acc = zeros(1,1);
%acc = zeros(1,vector_length);
acc(1) = 0;
%Acceleration:
jerk = zeros(1,1);
jerk(1) =sign*j;

for i = 1:j_t_discrete
    tmp_acc = acc(end) + sign*acc_inc;
    acc = [acc tmp_acc];
    tmp_spd = spd_i+sum(acc)*dt;
    tmp_j = sign*j;
    tmp_pos= pos_i+sum(spd)*dt;

    spd = [spd tmp_spd]
    pos = [pos tmp_pos];
    jerk = [jerk tmp_j];
end
if(skip_sacel == 0)
    %Constant acel
    for i = 1:cte_acc_pos_discrete
        tmp_acc = acc(end);
        tmp_spd = spd_i+sum(acc)*dt;
        tmp_j = 0;
        tmp_pos = pos_i+sum(spd)*dt;

        spd = [spd tmp_spd];
        pos = [pos tmp_pos];
        acc = [acc tmp_acc];
        jerk = [jerk tmp_j];
    end
end
%Negative Acceleration:
for i = 1:j_t_discrete
    tmp_acc = acc(end) - sign*acc_inc;
    tmp_spd = spd_i+sum(acc)*dt;
    tmp_j = -(sign*j);
    tmp_pos = pos_i + sum(spd)*dt;

    spd = [spd tmp_spd];
    pos = [pos tmp_pos];
    acc = [acc tmp_acc];
    jerk = [jerk tmp_j];
end

for i = 1:cte_spd_pos_discrete
    tmp_acc = 0;
    %tmp_spd = spd(end);
    tmp_spd = spd_max;
    tmp_j = 0;
    tmp_pos = pos_i + sum(spd)*dt;

    spd = [spd tmp_spd];
    pos = [pos tmp_pos];
    acc = [acc tmp_acc];
```

```
        jerk = [jerk tmp_j];
    end

    for i = 1:j_t_discrete
        tmp_acc = acc(end) + (-sign*acc_inc);
        acc = [acc tmp_acc];
        tmp_spd = spd_i+sum(acc)*dt;
        tmp_j = -sign*j;
        tmp_pos= pos_i+sum(spd)*dt;

        spd = [spd tmp_spd]
        pos = [pos tmp_pos];
        jerk = [jerk tmp_j];
    end
    if(skip_sacel == 0)
        %Constant acel
        for i = 1:cte_acc_pos_discrete
            tmp_acc = acc(end);
            tmp_spd = spd_i+sum(acc)*dt;
            tmp_j = 0;
            tmp_pos = pos_i+sum(spd)*dt;

            spd = [spd tmp_spd];
            pos = [pos tmp_pos];
            acc = [acc tmp_acc];
            jerk = [jerk tmp_j];
        end
    end
    %Negative Acceleration:
    for i = 1:j_t_discrete
        tmp_acc = acc(end) - (-sign*acc_inc);
        tmp_spd = spd_i+sum(acc)*dt;
        tmp_j = -(-sign*j);
        tmp_pos = pos_i + sum(spd)*dt;

        spd = [spd tmp_spd];
        pos = [pos tmp_pos];
        acc = [acc tmp_acc];
        jerk = [jerk tmp_j];
    end
end
```

Contenido Anexo 2. Hojas de características

A2.1. Hoja de características del motor de corriente continua 2

A2.2. Hoja de características del encoder óptico incremental 4

A2.3. Hoja de características del bloque reductor para motor 5

A2.4. Hoja de características de la etapa de potencia 6

A2.5. Hoja de características del kit de desarrollo STM32F429-Discovery 9

A2.1. Hoja de características del motor de corriente continua

DC040B Series Brush DC Motor

9233E871 19.1 VDC

The DC040B series brush commutated DC motor is a 40 mm diameter unit offered in 6 lengths with continuous output torques of 0.017 to 0.081 Nm.

Features:

- Speeds up to 8,000 RPM possible
- DC bus voltage up to 48 Vdc
- Eight standard windings - Special windings available
- 2 pole stator with ceramic magnets
- 7 slot skewed armature cogging reduction
- Sintered bronze bearings - ball bearings available
- Copper graphite brushed - RFI suppression available

Assembly Options:

- Encoder: E30A/B, E30C/D
- Gearbox: G30A, G40A, PLG42S, G51A
- Brake: B30A, B49

DC040B Series Salient Characteristics

MOTOR DATA	SYMBOL	UNITS	DC040B-1	DC040B-2	DC040B-3	DC040B-4	DC040B-5	DC040B-6
Max DC Terminal Voltage	V_T	V	48	48	48	48	48	48
Max Speed (Mechanical)	ω_{MAX}	rpm	8000	8000	8000	8000	7000	7000
Continuous Stall Torque[1]	T_{cs}	Nm	0.017	0.033	0.043	0.049	0.067	0.081
		oz-in	2.4	4.7	6.1	6.9	9.5	12
Peak Torque (Maximum)[1]	T_{pk}	Nm	0.086	0.20	0.26	0.32	0.40	0.50
		oz-in	12	28	37	45	56	71
Coulomb Friction Torque	T_f	Nm	0.0035	0.0042	0.0042	0.0046	0.0056	0.0056
		oz-in	0.50	0.60	0.60	0.65	0.80	0.80
Viscous Damping Factor	D	Nm s/rad	1.8E-06	2.3E-06	2.6E-06	3.0E-06	3.5E-06	3.7E-06
		oz-in/krpm	0.028	0.034	0.039	0.045	0.053	0.055
Thermal Time Constant	τ_{th}	min	7.2	11	12	13	14	14
Thermal Resistance	R_{th}	°C/W	23	19	17	15	14	11
Max. Winding Temperature	Θ_{MAX}	°C	155	155	155	155	155	155
Rotor Inertia	J_r	kg m^2	1.9E-06	3.2E-06	4.2E-06	5.6E-06	7.1E-06	8.5E-06
		oz-in-s^2	2.7E-04	4.6E-04	5.9E-04	7.9E-04	0.0010	0.0012
Motor Weight	W_m	g	200	250	290	340	390	440
		oz	7.0	8.9	10	12	14	16

OPTIONS		
Gearboxes	Encoders	Brakes
G30A, G40A, PLG42S, G51A	E30A/B, E30C/D	B30A, B49A

[1] Recorded at maximum winding temperature at 25°C ambient and without heatsink.

Figura 1. Características mecánicas de la serie DC040B

DC040B-1 through DC040B-6: Dimensional Drawings

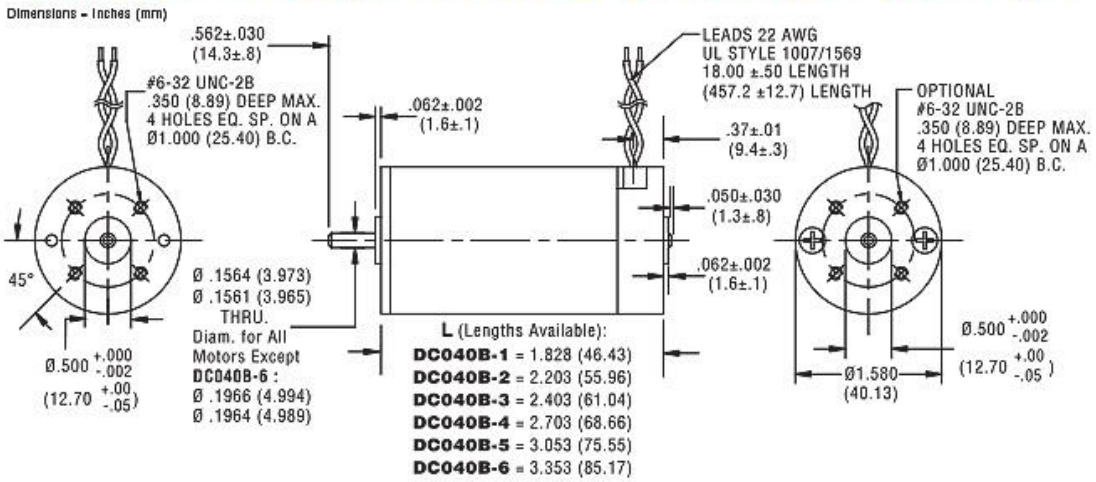


Figura 2. Descripción dimensional de la serie DC040B

DC040B-2: Performance Data

MOTOR DATA	SYMBOL	UNITS								
Rated Voltage **V1**	V_r	V	9.55	12.0	15.2	19.1	24.0	30.3	38.2	48.0
Rated Torque [1] ●	T_r	Nm	0.026	0.025	0.024	0.024	0.024	0.023	0.023	0.023
		oz-in	3.6	3.5	3.4	3.4	3.3	3.3	3.3	3.3
Rated Speed [1]	ω_r	rpm	5000	5100	5300	5310	5350	5360	5390	5390
Rated Current [1]	I_r	A	2.4	1.8	1.4	1.1	0.89	0.70	0.55	0.44
Rated Power [1]	P_r	W	13	13	13	13	13	13	13	13
No Load Speed	ω_{nl}	rpm	5870	5810	5920	5870	5880	5860	5880	5880
No Load Current	I_{nl}	A	0.38	0.30	0.25	0.19	0.16	0.12	0.095	0.076
Rated Voltage **V2**	V_r	V	7.58	9.55	12.0	15.2	19.1	24.0	30.3	38.2
Rated Torque [1] ●	T_r	Nm	0.028	0.028	0.027	0.027	0.027	0.027	0.027	0.027
		oz-in	4.0	3.9	3.9	3.8	3.8	3.8	3.8	3.8
Rated Speed [1]	ω_r	rpm	3440	3570	3680	3750	3790	3780	3810	3830
Rated Current [1]	I_r	A	2.6	2.0	1.6	1.2	0.98	0.77	0.61	0.49
Rated Power [1]	P_r	W	10	10	11	11	11	11	11	11
No Load Speed	ω_{nl}	rpm	4630	4600	4650	4650	4660	4620	4640	4660
No Load Current	I_{nl}	A	0.36	0.29	0.23	0.18	0.15	0.12	0.090	0.072
Motor Constant	K_M	Nm/√W	0.017	0.018	0.018	0.019	0.019	0.019	0.019	0.019
		oz-in/√W	2.5	2.6	2.6	2.6	2.7	2.7	2.7	2.7
Torque Constant	K_T	Nm/A	0.0148	0.0188	0.0234	0.0297	0.0372	0.0472	0.0593	0.0746
		oz-in/A	2.10	2.66	3.31	4.21	5.27	6.68	8.40	10.6
Voltage Constant	K_E	V s/rad	0.0148	0.0188	0.0234	0.0297	0.0372	0.0472	0.0593	0.0746
		V/krpm	1.55	1.97	2.45	3.11	3.90	4.94	6.21	7.81
Terminal Resistance	R_{mt}	Ω	0.720	1.08	1.63	2.53	3.94	6.21	9.78	15.4
Inductance	L	mH	0.52	0.84	1.3	2.1	3.3	5.3	8.3	13
Peak Current	I_{pk}	A	13	11	9.3	7.5	6.1	4.9	3.9	3.1
Electrical Time Constant	τ_e	ms	0.72	0.78	0.79	0.82	0.84	0.85	0.85	0.85
Mechanical Time Constant	τ_m	ms	11	9.9	9.7	9.3	9.2	9.1	9.0	9.0

[1] Recorded at maximum winding temperature at 25°C ambient and without heatsink.

Figura 3. Características eléctricas de la serie DC040B

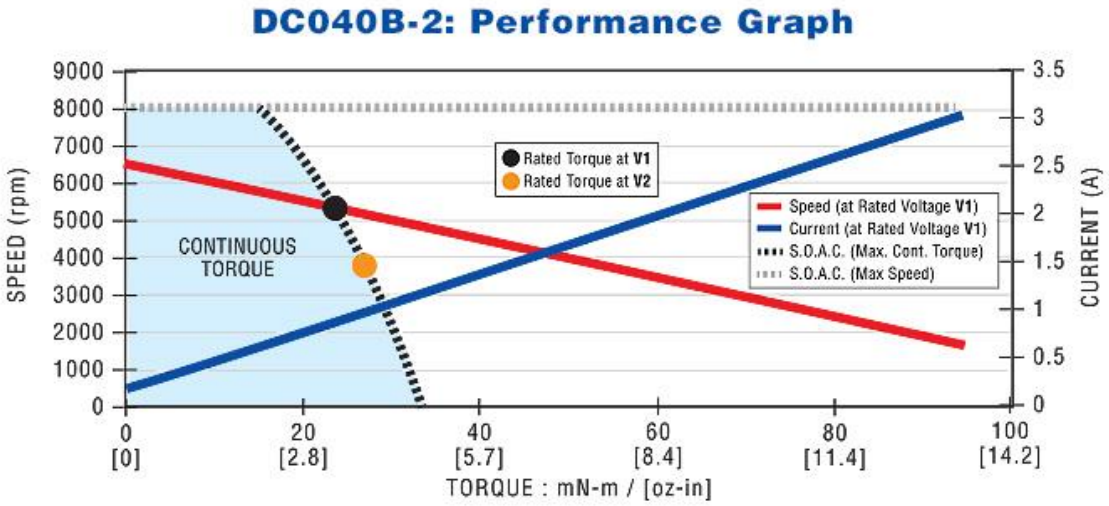


Figura 4. Gráfico de funcionamiento del motor DC040B-2

S.O.A.C. = Safe Operating Area for Continuous duty

A2.2. Hoja de características del encoder óptico incremental

E30 Incremental Optical Encoder

Cost effective feedback devices for positioning applications. Encoder technologies include transmissive optical and reflective optical. For maximum EMI and RFI noise immunity, differential line drivers with complementary outputs are available with most models as an optional configuration.

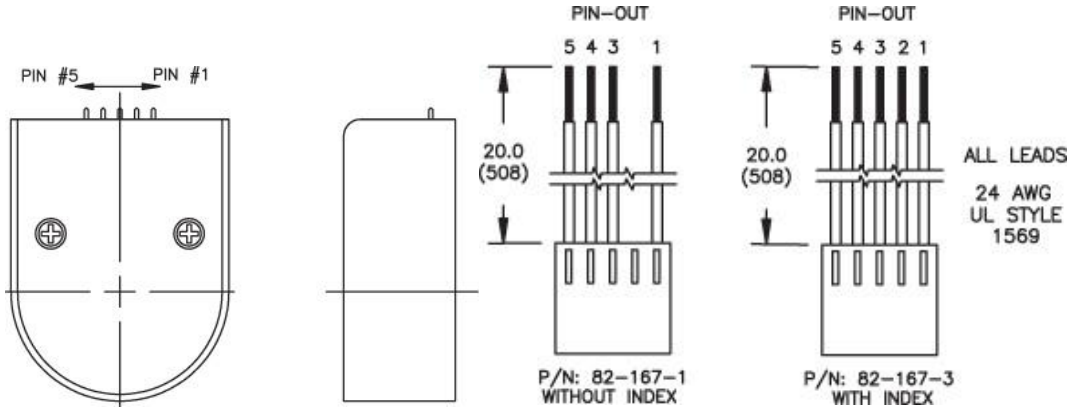


Figura 5. Identificación de los terminales del E30

E30A Encoder Data	Symbol	Units	Value
Available Resolutions	-	-	100, 200, 256, 360, 500, 512, 1000, 1024
Output	-	-	2-Channel Quadrature Output
Output Interface	-	-	TTL Compatible
Supply Voltage	V_{cc}	VDC	4.5 to 5.5
Supply Current	I_{cc}	mA	40 max
High Level Output Voltage	V_{OH}	V	2.4 min
Low Level Output Voltage	V_{OL}	V	0.4 max
Maximum Operating Frequency	f_{max}	kHz	100
Maximum Operating Temperature	θ_{max}	°C	-40 to +100
Encoder Weight (Mass)	w_E	oz	2.0
		g	57

E30A Connection Chart		
Pin	Color	Function
1	Black	Ground
2	-	No Connection
3	Yellow	Channel A
4	Red	Vcc
5	Blue	Channel B

Options
- Differential line drive with complementary outputs.
- Alternate leadwire assemblies.

Notes:
[1] Motor ball bearings are required for optimal performance.

Figura 6. Características técnicas del *encoder* incremental E30A

A2.3. Hoja de características del bloque reductor para motor

G30A Planetary Gearbox

G30A 24:1

Pittman planetary gears are perfect for servo applications where maximum efficiency, minimum backlash, and a smaller mechanical footprint are necessary. In contrast to spur gears, planetary gears share the load through a pinion (sun gear) over multiple mating gears (planet gears). Planetary gears are offered with plastic or metal gears and sleeve or ball bearing outputs.

		Reduction Ratio Designation													
Specification	Units	G30A 4:1	G30A 5:1	G30A 6:1	G30A 16:1	G30A 24:1	G30A 36:1	G30A 64:1	G30A 96:1	G30A 144:1	G30A 216:1	G30A 256:1	G30A 384:1	G30A 576:1	G30A 1296:1
Maximum Load	oz-in	350	350	350	500	500	500	920	920	920	920	1250	1250	1250	1250
	Nm	2.47	2.47	2.47	3.53	3.53	3.53	6.5	6.5	6.5	6.5	8.83	8.83	8.83	8.83
Weight (Mass)	oz	3.9	3.9	3.9	4.5	4.4	4.4	4.9	4.9	4.9	4.9	5.5	5.5	5.5	5.5
	g	110.6	110.6	110.6	127.6	124.7	124.7	138.9	138.9	138.9	138.9	155.9	155.9	155.9	155.9
Length (L)	Inches	1.325	1.325	1.325	1.555	1.555	1.555	1.785	1.785	1.785	1.785	2.015	2.015	2.015	2.015
	mm	33.7	33.7	33.7	39.5	39.5	39.5	45.3	45.3	45.3	45.3	51.2	51.2	51.2	51.2
Exact Ratio	-	4/1	5/1	6/1	16/1	24/1	36/1	64/1	96/1	144/1	216/1	256/1	384/1	576/1	1296/1
Efficiency	-	0.90	0.90	0.90	0.81	0.81	0.81	0.73	0.73	0.73	0.73	0.66	0.66	0.66	66
Shaft Rotation	-	CW	CW	CW	CW	CW	CW	CW	CW	CW	CW	CW	CW	CW	CW

Standard Construction
- Sintered steel gears for high torque capacity and low audible noise
- Sintered metal output bearing

Options
- Alternate mounting and shaft configurations
- Additional ratios available
- Output ball bearings for high radial loads

NOTES:

[1]Maximum load represents gearbox capability only. Continuous load torque capability will vary with gear ratio, motor selection, and operating conditions.

[2]Shaft rotation is designated while looking at output shaft with motor operating in a clockwise direction. Gearboxes have bi-directional capability.

[3]Flange option (G30AF) adds 31 grams to gearbox weight.

Figura 7. Características técnicas del bloque reductor G30A

A2.4. Hoja de características de la etapa de potencia

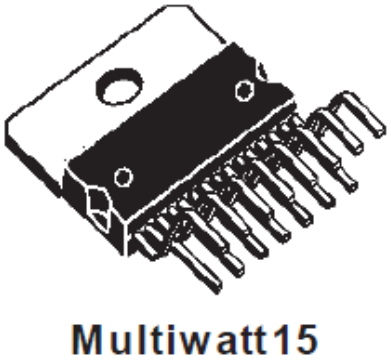
L298 DUAL FULL-BRIDGE DRIVER

The L298 is an integrated monolithic circuit in a 15-lead Multiwatt and PowerSO20 packages. It is a high voltage, high current dual full-bridge driver designed to accept standard TTL logic levels and drive inductive loads such as relays, solenoids, DC and stepping motors. Two enableinputs are provided to enableor disable the device independently of the input signals. The emitters of the lower transistors of each bridge are connected together and the corresponding external terminal can be used for the connection of anexternal sensing resistor. An additional supply input is provided so that the logic works at a lower voltage.

PIN FUNCTIONS (refer to the block diagram)

MW.15	PowerSO	Name	Function
1;15	2;19	Sense A; Sense B	Between this pin and ground is connected the sense resistor to control the current of the load.
2;3	4;5	Out 1; Out 2	Outputs of the Bridge A; the current that flows through the load connected between these two pins is monitored at pin 1.
4	6	V_S	Supply Voltage for the Power Output Stages. A non-inductive 100nF capacitor must be connected between this pin and ground.
5;7	7;9	Input 1; Input 2	TTL Compatible Inputs of the Bridge A.
6;11	8;14	Enable A; Enable B	TTL Compatible Enable Input: the L state disables the bridge A (enable A) and/or the bridge B (enable B).
8	1,10,11,20	GND	Ground.
9	12	VSS	Supply Voltage for the Logic Blocks. A100nF capacitor must be connected between this pin and ground.
10; 12	13;15	Input 3; Input 4	TTL Compatible Inputs of the Bridge B.
13; 14	16;17	Out 3; Out 4	Outputs of the Bridge B. The current that flows through the load connected between these two pins is monitored at pin 15.
–	3;18	N.C.	Not Connected

Figura 8. Funciones de los pines del integrado L298



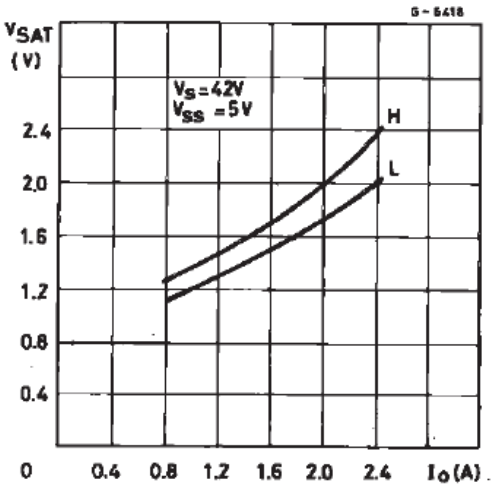


Figura 9. Encapsulado (izq.) y curva de tensión de saturación-corriente de salida (der.) del L298

ELECTRICAL CHARACTERISTICS (V_S = 42V; V_{SS} = 5V, T_j = 25°C; unless otherwise specified)

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
V_S	Supply Voltage (pin 4)	Operative Condition	V_{IH} +2.5		46	V
V_{SS}	Logic Supply Voltage (pin 9)		4.5	5	7	V
I_S	Quiescent Supply Current (pin 4)	V_{en} = H; I_L = 0 V_i = L V_i = H		13 50	22 70	mA mA
		V_{en} = L V_i = X			4	mA
I_{SS}	Quiescent Current from V_{SS} (pin 9)	V_{en} = H; I_L = 0 V_i = L V_i = H		24 7	36 12	mA mA
		V_{en} = L V_i = X			6	mA
V_{iL}	Input Low Voltage (pins 5, 7, 10, 12)		–0.3		1.5	V
V_{iH}	Input High Voltage (pins 5, 7, 10, 12)		2.3		VSS	V
I_{iL}	Low Voltage Input Current (pins 5, 7, 10, 12)	V_i = L			–10	µA
I_{iH}	High Voltage Input Current (pins 5, 7, 10, 12)	Vi = H ≤ V_{SS} –0.6V		30	100	µA
V_{en} = L	Enable Low Voltage (pins 6, 11)		–0.3		1.5	V
V_{en} = H	Enable High Voltage (pins 6, 11)		2.3		V_{SS}	V
I_{en} = L	Low Voltage Enable Current (pins 6, 11)	V_{en} = L			–10	µA
I_{en} = H	High Voltage Enable Current (pins 6, 11)	V_{en} = H ≤ V_{SS} –0.6V		30	100	µA
$V_{CEsat(H)}$	Source Saturation Voltage	I_L = 1A I_L = 2A	0.95	1.35 2	1.7 2.7	V V
$V_{CEsat(L)}$	Sink Saturation Voltage	I_L = 1A (5) I_L = 2A (5)	0.85	1.2 1.7	1.6 2.3	V V
V_{CEsat}	Total Drop	I_L = 1A (5) I_L = 2A (5)	1.80		3.2 4.9	V V
V_{sens}	Sensing Voltage (pins 1, 15)		–1 (1)		2	V

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
T_1 (V_i)	Source Current Turn-off Delay	0.5 V_i to 0.9 I_L (2); (4)		1.5		µs
T_2 (V_i)	Source Current Fall Time	0.9 I_L to 0.1 I_L (2); (4)		0.2		µs
T_3 (V_i)	Source Current Turn-on Delay	0.5 V_i to 0.1 I_L (2); (4)		2		µs
T_4 (V_i)	Source Current Rise Time	0.1 I_L to 0.9 I_L (2); (4)		0.7		µs

Figura 10. Características eléctricas del L298

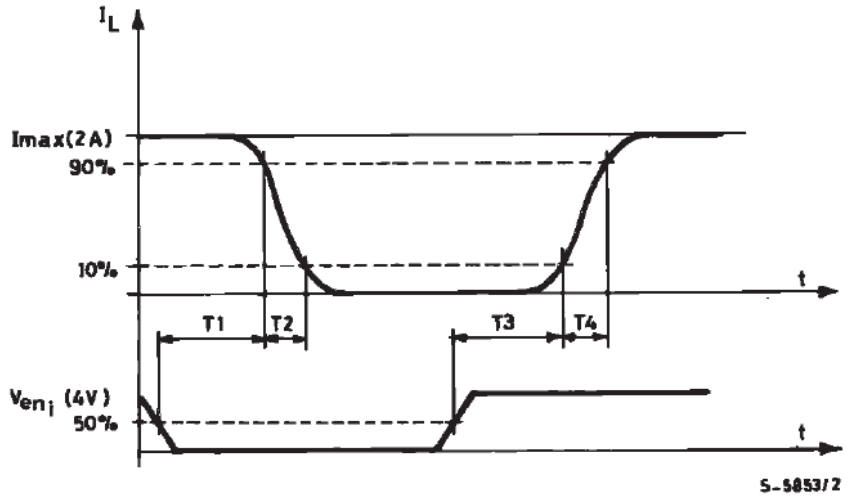


Figura 11. Curva de conmutación del L298

L298

ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_S	Power Supply	50	V
V_{SS}	Logic Supply Voltage	7	V
V_I, V_{en}	Input and Enable Voltage	–0.3 to 7	V
I_O	Peak Output Current (each Channel) – Non Repetitive (t = 100µs) –Repetitive (80% on –20% off; t_{on} = 10ms) –DC Operation	 3 2.5 2	 A A A
V_{sens}	Sensing Voltage	–1 to 2.3	V
P_{tot}	Total Power Dissipation (T_{case} = 75°C)	25	W
T_{op}	Junction Operating Temperature	–25 to 130	°C
T_{stg}, T_j	Storage and Junction Temperature	–40 to 150	°C

PIN CONNECTIONS (top view)

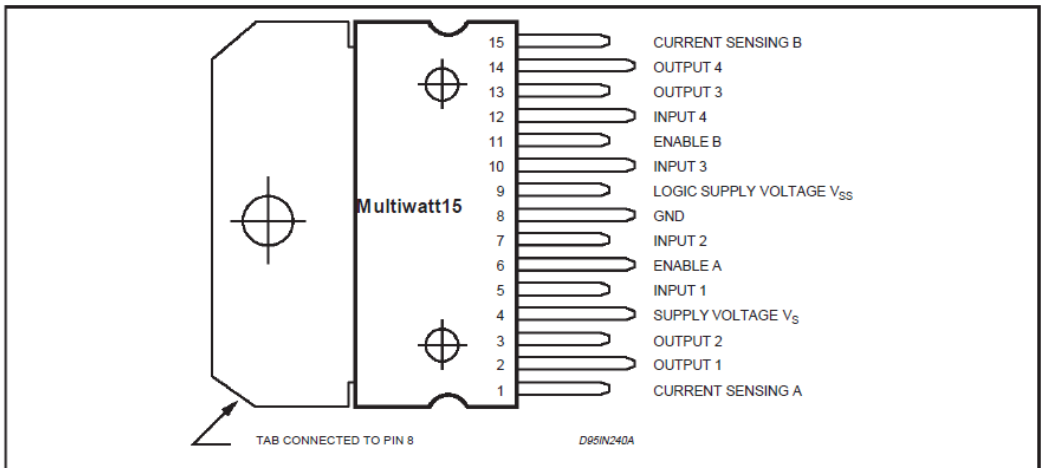
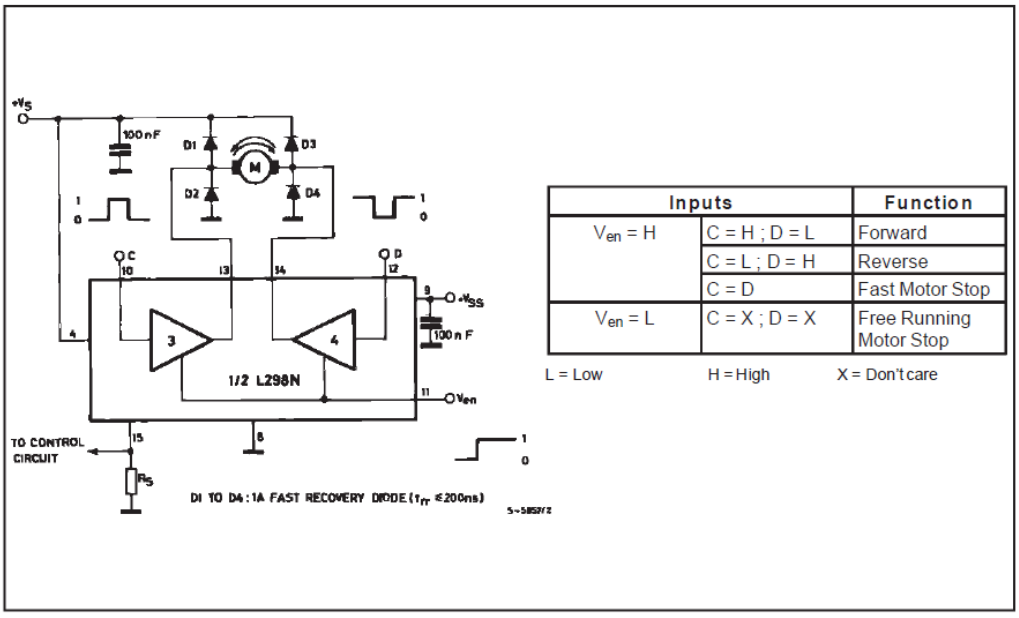


Figura 12. Valores máximos de funcionamiento (arriba) e identificación de los pines (abajo) del L298

Figure 6 : Bidirectional DC Motor Control.



Inputs		Function
V_{en} = H	C = H ; D = L	Forward
	C = L ; D = H	Reverse
	C = D	Fast Motor Stop
V_{en} = L	C = X ; D = X	Free Running Motor Stop

L = Low H = High X = Don't care

Figura 13. Descripción del funcionamiento para el control de un motor CC del L298

A2.5. Hoja de características del kit de desarrollo STM32F429-Discovery

STM32F429-Discovery development kit

Symbol	Parameter	Conditions[1]	Min	Typ	Max	Unit
V_{DD}	Standard operating voltage		1.7[2]	-	3.6	V
V_{DDA}[3][4]	Analog operating voltage (ADC limited to 1.2 M samples)	Must be the same potential as V_{DD}[5]	1.7[2]	-	2.4	
	Analog operating voltage (ADC limited to 2.4 M samples)		2.4	-	3.6	
V_{BAT}	Backup operating voltage		1.65	-	3.6	
V_{12}	Regulator ON: 1.2 V internal voltage on V_{CAP_1}/V_{CAP_2} pins	Power Scale 3 ((VOS[1:0] bits in PWR_CR register = 0x01), 120 MHz HCLK max frequency	1.08	1.14	1.20	
		Power Scale 2 ((VOS[1:0] bits in PWR_CR register = 0x10), 144 MHz HCLK max frequency with over-drive OFF or 168 MHz with over-drive ON	1.20	1.26	1.32	
		Power Scale 1 ((VOS[1:0] bits in PWR_CR register = 0x11), 168 MHz HCLK max frequency with over-drive OFF or 180 MHz with over-drive ON	1.26	1.32	1.40	
	Regulator OFF: 1.2 V external voltage must be supplied from external regulator on V_{CAP_1}/V_{CAP_2} pins[6]	Max frequency 120 MHz	1.10	1.14	1.20	
		Max frequency 144 MHz	1.20	1.26	1.32	
		Max frequency 168 MHz	1.26	1.32	1.38	
V_{IN}	Input voltage on RST and FT pins[7]	2 V ≤ V_{DD} ≤ 3.6 V	− 0.3	-	5.5	V
		V_{DD} ≤ 2 V	− 0.3	-	5.2	
	Input voltage on TTa pins		− 0.3	-	V_{DDA}+ 0.3	
	Input voltage on BOOT0 pin		0	-	9	
P_D	Power dissipation at T_A = 85 °C for suffix 6 or T_A = 105 °C for suffix 7[8]	LQFP100	-	-	465	mW
		WLCSP143	-	-	641	
		LQFP144	-	-	500	
		UFBGA169	-	-	385	
		LQFP176	-	-	526	
		UFBGA176	-	-	513	
		LQFP208	-	-	1053	
		TFBGA216	-	-	690	
T_A	Ambient temperature for 6 suffix version	Maximum power dissipation	− 40		85	°C
		Low power dissipation[9]	− 40		105	
	Ambient temperature for 7 suffix version	Maximum power dissipation	− 40		105	°C
		Low power dissipation[9]	− 40		125	

Figura 14. Condiciones generales de funcionamiento STM32F429-Discovery

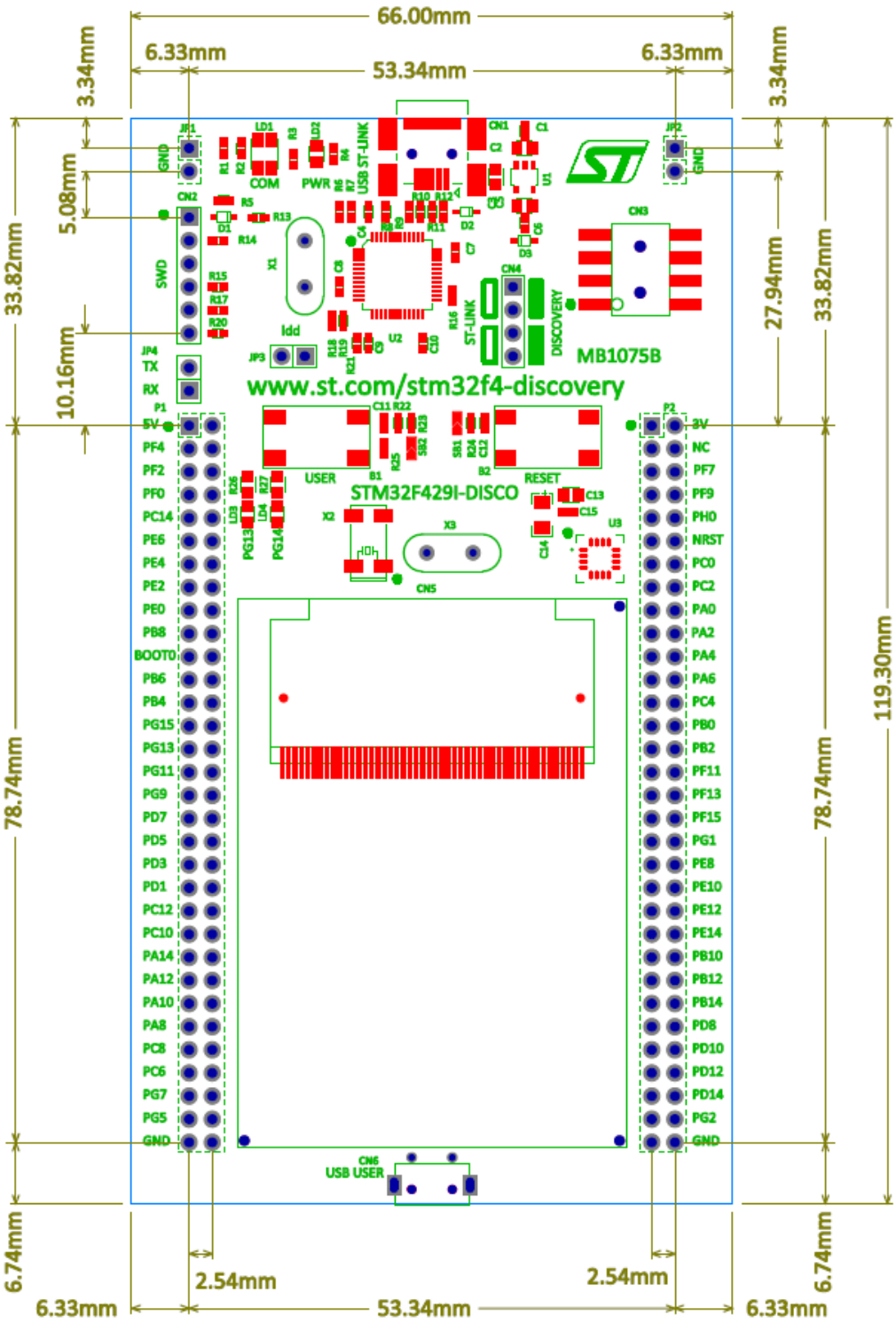


Figura 15. Dimensiones y disposición de los componentes en el kit STM32F429-Discovery

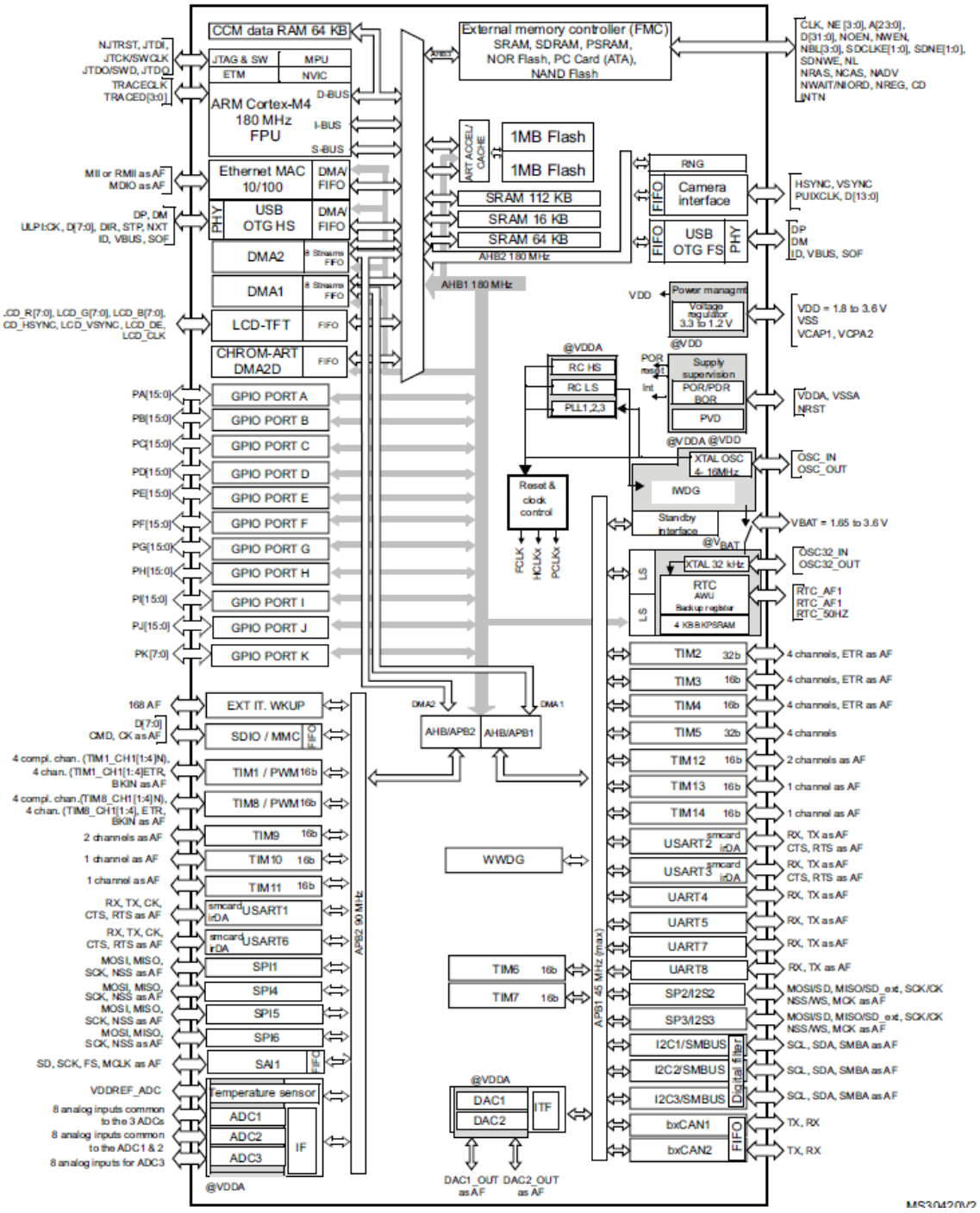


Figura 16. Diagrama de bloques de los periféricos del kit STM32F429-Discovery